COMPUTING AND DATA VISUALISATION FOR SELECTED PROBLEMS IN SOLAR ENERGY USING PYTHON PROGRAMMING LANGUAGE

BY

MISHECK LUNGU

A dissertation submitted to the University of Zambia in partial fulfillment of the requirements for the award of the degree of Master of Science in Physics

The University of Zambia

March, 2024

COPYRIGHT

DECLARATION

I, Misheck Lungu, do hereby declare that this dissertation represents my work, and that it has not previously been submitted for any degree or academic qualification. Except where otherwise indicated by reference or acknowledgment within the dissertation.

Signature: ……. Date: …….

DEDICATION

This research is dedicated to my wonderful parents for always supporting me in my studies. I also want to thank my brothers and sisters for believing in me and making my life happier. And a big thank you to my nephews and nieces who motivate me to work hard and never give up on my goals. I hope this research shows how much I love my family and how they've helped me succeed.

CERTIFICATE OF APPROVAL

This dissertation of Misheck Lungu has been approved as a partial fulfillment of the requirements for the award of the degree of Master of Science in Physics by the University of Zambia.

Examiner 1: ……. Signature: ……. Date: ……

Examiner 2: ……. Signature: ……. Date: ……

Examiner 3: ……. Signature: ……. Date: ……

Chairperson of Viva Voce: ……... Signature: ……. Date: ……

Principal Supervisor: ……. Signature: ……. Date: ……

Co-supervisor: ……Signature: ……. Date: …….

ABSTRACT

Solar energy plays a vital role in reducing energy deficits by providing a renewable and sustainable power source, and it significantly reduces contributions to climate change compared to fossil fuels. A deeper understanding of the concepts involved in the study of solar energy is crucial for the development of more efficient systems. Many textbooks contain key concepts such as mathematical expressions, tabulated values, and graphical representations that are used as pedagogical tools for the mastery of the subject. However, many of these tools are not interactive and may not be able to provide solutions in real-time. Therefore, this study addresses this gap by developing a user-friendly graphical interface for solving, simulating and visualizing six key solar energy problems. The codes were created using Python 3 and its libraries from the Jupyter Notebook environment. To empower user interaction and variable control, Ipywidgets, a Jupyter Notebook library was used. Later, the codes were migrated to the Tkinter framework for further development. The developed graphical user interface (GUI) applications were converted into executable files which were deployed on Windows systems. By facilitating real-time problem solving and data visualization the applets can enhance learning and empower researchers.

Keywords: *Solar Energy, Energy Deficits, Python, Data Visualization, Graphical User Interface, Executable Files*

ACKNOWLEDGEMENT

I wish to express my sincere gratitude to my principal supervisor Professor Prem Jain and co-supervisor Mr. Gift Sichone for the guidance and constructive academic advice rendered during my research. Sincere gratitude to the lecturers and staff in the Department of Physics for their encouragement, and to the management of the School of Natural Sciences for the well-organized master's programs. I also extend my gratitude to my family for their moral support.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

There is a rapid growth in the world's energy needs. However, the energy sector is a major source of greenhouse gases whose emissions have resulted in climate change which is one of the most important global agenda today [1]. A way to mitigate climate change is a rapid shift to renewable sources of energy which necessitates a constant stream of trained manpower in solar and other renewable energy technologies. However, despite the advancement in the energy sector, much remains to be done to meet the ambitious targets set forth in the international agreements such as the Paris Agreement [2].

According to the Paris Agreement, solar energy plays a vital role in mitigating climate change by significantly reducing greenhouse gas emissions. It is recognized as a clean and renewable energy source that supports the transition away from fossil fuels, aligning with the agreement's goal of limiting global warming to well below 2 degrees Celsius above pre-industrial levels, with efforts to limit it to 1.5 degrees Celsius. Solar power not only contributes to decarbonizing the energy sector but also enhances resilience to climate change impacts by diversifying energy sources and decentralizing production, as emphasized by the Paris Agreement. Furthermore, solar energy promotes sustainable development by providing clean and affordable energy access, particularly in underserved areas, thus contributing to poverty alleviation and economic growth. The Paris Agreement underscores the importance of global collaboration and partnership in achieving its objectives, and solar energy presents opportunities for international cooperation, technology transfer, and capacity building to accelerate the transition to a low-carbon economy, fostering a more sustainable and resilient future for all [2].

In the realm of educational and research in solar energy, there is need for creating computing and data visualization tools capable of providing real-time data processing and analysis. These tools need to offer user-friendly interfaces that will facilitate quick and efficient learning experiences.

1.2 Statement of the Problem

While textbooks in solar energy physics serve as valuable resources, they sometimes fall short in providing a comprehensive understanding of the involved concepts due to limitations in demonstration capability and maneuverability. Similarly, solar energy labs may face challenges in accessing adequate equipment as pedagogical tools for effective demonstrations. Consequently, there is a need for additional resources to enhance the depth of understanding of the concepts involved in solar energy physics and engineering.

1.3 Aim

The primary aim of this study was to develop interactive computer programs and visualization tools using the Python programming language to solve specific problems in solar energy. These tools will address six problems, namely: blackbody radiation curves, determining daylengths or daylight hours for a specific geographical location throughout the year, calculating daily solar declination and eccentricity correction factor, estimating hourly, daily, and monthly solar extraterrestrial irradiance for a given geographic location, and analyzing reflectance and transmittance at dielectric interfaces i.e. air-glass interface using Fresnel's equations. These problems were selected based on their pedagogical value. The tools developed aim to provide students and researchers with enhanced learning experiences and insights into the principles of solar energy.

1.4 Objectives

(i) Create and develop prototype Python based computer codes for solving each of the selected problems.

(ii) Create Python based applets for simulating and visualizing.

(iii) Create a Python based standalone application with the menu system for accessing the applets.

1.5 Research Questions

(i) How can the understanding of solar energy physics concepts be enhanced?

(ii) What methodologies can be employed to integrate real-time data by developing computer programs, thereby reducing reliance on pre-processed data?

(iii) What computational tools can be developed to optimize the efficiency of solving analytical mathematical problems in solar energy physics, thereby reducing computational time while maintaining accuracy?

1.6 Significance of the Study

The developed tools have the potential to benefit solar energy applets. Textbooks, despite serving as essential educational tools are limited in explaining important concepts. Many textbooks are static and may not have the dynamic and interactive features that are necessary understanding difficult concepts easily. In order to mitigate this challenge and keep pace with the ever-changing technological world, supplementally tools are needed. In institutions where access to sophisticated laboratory equipment is limited the developed tools can compensate for this limitation by functioning as pedagogical bridge. The tools can enhance the students/researchers understanding of the subject matter regardless of the geographical location or resource constraints. They enable real-time calculation, visualization, and simulation of solar energy principles, relying on established analytical expressions.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Prof. Prem Jain identified and selected the six problems addressed in this dissertation and suggested to the author (written notes, 2021). An internet-based literature review identified Java-based web applets and web services for blackbody curves, daylight hours calculation, extraterrestrial irradiation, and Fresnel reflection and transmission calculation. However, no applets were found specifically designed for solar declination and the eccentricity correction factor. This gap highlighted the need for additional research and development in these areas to support solar energy education.

2.2 Blackbody Radiation

Among the blackbody simulation curve applets, the PhET Simulations Blackbody Curves and Thermal Radiation Curves applets were identified. The PhET Simulation Blackbody Curves applet plots the blackbody spectrum for absolute temperatures ranging from 200K to 10000K using a temperature slider, annotating the peak wavelength and intensity. It also visually demonstrates how the volume of the blackbody varies with absolute temperature [3]. The Thermal Radiation Curves applet plots blackbody radiation curves for preselected temperatures or allows users to input their desired temperature [4]. However, these two blackbody applets lack verification for Wien's Displacement Law or Stefan-Boltzmann's Law.

The third applet that was found online was the Blackbody Calculator. This calculator has two columns, an input where the user enters the following variables: temperature, emissivity, wavelength and recursion velocity, and the results column where radiance emittance, radiance, peak spectral radiance and wavelength peak are shown [5]. However, it lacks a graphical comparison of the peak wavelengths as the temperature increases or decreases.

Another applet that was found is the Blackbody Radiation Spectrum model which shows six fixed-temperature curves between maximum and minimum temperature and, a red variable-temperature curve that can be adjusted using a slider [6]. However, a comparison of the peak wavelengths obtained from Wien's law with the graphical values is not addressed.

There was no applet found that specifically dealt with calculating and visualizing the blackbody fractions.

2.3 Daylength

The Daylight Explorer applet calculates the hours of daylight received for a particular day during the year at a given latitude. It also enables users to change the latitude of a particular location and the day of the year using sliders, and it can calculate the average annual daylight hours [7]. However, it cannot generate multiple daylight plots for different geographical locations.

The Sun Calculator applet provides detailed information about sunrise, sunset, and twilight times for any location worldwide. It also calculates solar noon, daylength, and the Sun's position and altitude. It is made in such a way that users can select a city or enter geographical coordinates to get the solar data they need. The tool's interactive features make it easy to visualize and understand solar patterns [8]. However, visualization of daylength of a particular location is not clear and it lacks comparison of multiple locations.

2.4 Extraterrestrial Radiation

The Solar Radiation data website offers a web-based service for extraterrestrial radiation and Top of the Atmosphere (TOA) allowing users to calculate daily extraterrestrial irradiance values [9]. However, this service lacks hourly and monthly values for extraterrestrial irradiation and does not provide visual analysis of the data.

Another tool that calculates the extraterrestrial radiation is the solar radiation calculator found at Santa Clara University website. This applet calculates extraterrestrial solar radiation on a horizontal surface at the top of the atmosphere. It allows users input latitude of their choice using the latitude entry box, then using the dropdown menus they can select the month and day. The output is displayed in watts per square meter (W/m^2) [10]. This tool is useful however, it does not cater for the calculations involving hourly extraterrestrial radiation and the visual as well as tabular analysis of the data is lacking.

2.5 Reflection and Refraction at Dielectric Interfaces

The Fresnel reflection and transmission calculator compute the power reflectivity and transmission of a plane wave at a dielectric interface using the Fresnel equations. The calculator also computes an average result corresponding to unpolarized light [11]. The limitation of this calculator is that

it does not provide visual analysis and plots of the parallel and perpendicular reflection coefficients, as well as the average reflection coefficients. Incorporating visual analysis would not only facilitate a deeper understanding of the results but also enhance the usability and interpretability of the calculator's output, particularly in educational or research contexts

2.6 Eccentricity Correction Factor and Solar Declination

There were no applets found for calculating solar declination and eccentricity correction factor.

In addition to the absence of applets for calculating solar declination and eccentricity correction factor, online resources specifically tailored for this purpose are notably not adequate. Despite the availability of graphs and tables in various books [12], there remains a gap in accessible digital resources for these calculations. This limitation hinders the convenience and efficiency of obtaining accurate values for solar declination and eccentricity correction factor, particularly in practical applications where real-time or interactive tools are preferred.

Textbooks contain equations which require manual solving without the benefit of an iterative process. Additionally, the reliance on pre-processed tables and graphs provided in textbooks can be limiting, as they lack interactivity and may not fully accommodate diverse user requirements or scenarios. Incorporating these limitations, there arises a need for enhanced online and offline resources that not only provide comprehensive datasets but also offer intuitive visualizations and interactive functionalities to facilitate deeper understanding and analysis of solar radiation dynamics.

CHAPTER 3

MATERIALS AND METHODS

3.1 Introduction

This chapter explains the procedures, methodologies and tools employed create applets. The first part involved identifying mathematical expressions relevant for solving each problem. The identified expressions were later translated into executable algorithms. This process involved breaking down these expressions into smaller components, selecting numerical methods and devising strategies on how to solve them. Then the algorithms were translated into computer codes. The codes were created using Python programming language, specifically Python 3 version and its libraries using the Jupyter Notebook environment. Python, chosen for its versatility and extensive library support, is an open-source, high-level, general-purpose programming language. Renowned for its simplicity, clarity, and adaptability, Python has emerged as a preferred choice for a wide range of applications, from web development to scientific computing [13]. It was created by Guido van Rossum and first released in 1991, placing a strong emphasis on the usability and readability of the code [14].

3.2 Environment

The codes were developed and executed in the Jupyter ecosystem, using JupyterNBClassic and JupyterLab. These environments facilitated code manipulation, visualization explanation and documentation [15]. Furthermore, Jupyter's interactive features enabled iterative development and exploration of data, enhancing the overall workflow efficiency. The writing of text and equations was done using Markdown, which provided a simple yet powerful markup language for structuring content, including mathematical expressions, thereby ensuring clarity and readability in the presentation of results.

3.3 Scientific and Numerical Computations

To perform numerical computation, two libraries were utilized: Numerical Python, commonly known as NumPy [16], and Scientific Python, popularly known as SciPy [17]. The former was employed for computing and data analysis using its constituents, while the latter was utilized for numerical integration. NumPy played a pivotal role in handling complex numerical operations and data analysis tasks, leveraging its extensive array-based functions and mathematical operations.

This enabled efficient computation and manipulation of arrays, facilitating a wide range of scientific computations with ease and speed. SciPy, known for its comprehensive collection of numerical algorithms and tools, was specifically employed for numerical integration tasks. By leveraging SciPy's rich set of integration routines, including methods for both definite and indefinite integration, precise numerical solutions could be obtained for various mathematical problems. Together, the combined capabilities of NumPy and SciPy empowered the computational framework with robust numerical computation capabilities, enabling the accurate and efficient solution of complex mathematical problems encountered in the project.

3.4 Graphical Analysis and Tables

For graphical representation and visualization, Matplotlib, a versatile Python plotting library renowned for its flexibility and extensive plotting capabilities was employed [18]. Additionally, the Pandas library which is known for its powerful data manipulation and analysis capabilities, was utilized to generate structured tables. The tables were organized and formatted before being exported to an Excel spreadsheet format [19]. By utilizing both Matplotlib and Pandas, the computational framework was equipped with comprehensive tools for visualizing data and presenting analytical results with clarity and precision.

3.5 Interactivity and Animation

The first version of all the computer programs were formulated to generate static graphs and tables which were later upgraded to interactive visuals, boasting user engagement and exploration capabilities. To enable user interaction and variable control, Ipywidgets [20], a Jupyter Notebook library, were utilized to create a graphical user interface. These widgets allow users to interact with the data and observe how different inputs affect the results. Buttons, dropdown menus, sliders, entry boxes and multiple select are among the widgets that were used. Buttons are interactive elements used to trigger actions or commands, such as submitting forms or initiating processes. Dropdown menus present users with a list of options, allowing them to select one from the available choices. Sliders enable users to select a value from a continuous range by dragging a handle along a track, commonly used for selecting numerical values within a specific range. Entry boxes, also known as text input fields, allow users to input text or numerical values directly into a field. Multiple select widgets enable users to choose multiple items from a list simultaneously, useful for selecting multiple options or categories. However, sharing and creating executable files

using Ipywidgets proved to be complex and raised compatibility issues. Therefore, the codes were changed to a different framework called tkinter. Tkinter is a standard Python library used to create graphical user interfaces (GUIs). It provides a set of tools and widgets for building windows, dialog boxes, buttons, menus, and other GUI components in Python applications [21]. These GUI applications were converted into executable files, facilitating easy distribution and deployment on Windows systems.

CHAPTER 4

RESULTS AND DATA ANALYSIS

4.1 Blackbody Radiation

4.1.1 Theory

A blackbody is a conceptual object that totally absorbs all incoming radiation while maintaining a thermal stable state, and subsequently releases all absorbed radiation. The released radiation is known as blackbody radiation. The term "blackbody" originates from the fact that at room temperature such an object would radiate minimal visible light, appearing black to the human eye. Although no real object perfectly fits the definition, many objects exhibit similar characteristics to a certain extent. Knowledge about the blackbody is important for various applications including the design of solar energy technologies.

In 1879 Austrian physicist Josef Stefan came up with a law which describes the relationship between the radiant power emitted by a blackbody and its absolute temperature. He asserted that the radiant power denoted as B_λ is proportional to the fourth power of the blackbody's absolute temperature T.

$$B_\lambda \propto T^4 \tag{4.1.1}$$

Ludwig Boltzmann later derived Josef Stefan's law using thermodynamic principles showing that; if B_λ is the radiant power emitted from a unit area in one second of a blackbody and T is the absolute temperature in degrees Kelvin, giving

$$B_\lambda(T) = \sigma T^4 \tag{4.1.2}$$

The proportionality constant σ is called the Stefan-Boltzmann constant which has the value $5.67 \times 10^8 \mathrm{W/m^2.K^2}$ [22].

Wilhelm Wien's law, was discovered a rule in 1893 that says that the maximum wavelength is inversely proportional to the absolute temperature of the body. The larger the absolute temperature of the body, the shorter the wavelength of light it emits. He found that the product $\lambda_m.\mathrm{T}$ is an absolute constant which gives 0.2898cm.K. However, for longer wavelengths such as microwaves and radio waves Wien's law does not hold. Because of this inadequacy Max Planck came up with a more general law that describes the distribution of radiation across all wavelengths [23].

In 1900 Maxwell Planck, a Germany physicist developed a mathematical relationship called Planck's radiation law that explains the spectral- energy distribution of radiation emitted by a blackbody. In his theory, Planck proposed that the sources of radiation consist of oscillating atoms, with each oscillator possessing vibrational energy levels existing in discrete values rather continuous. He further postulated that the transition of an oscillator from a higher energy state E_1 to a lower energy state E_2 results in the emission of discrete energy quanta represented by the equation $E_1 - E_2$, known as photons. This transition is quantified by the equation $E_1 - E_2 = hv$, where h represents Planck's constant and v denotes the frequency of the radiation. Planck's law for the energy $E\lambda$ radiated per unit volume by a cavity of a blackbody in the wavelength interval λ to $\lambda + \delta\lambda$ can be written in terms of Planck's constant (h), the speed of light (c), the Boltzmann's constant (k), and the absolute temperature (T):

$$E_\lambda = \frac{8\pi hc}{\lambda^5} \times \frac{1}{\left(e^{\left(\frac{hc}{KT\lambda}\right)} - 1\right)} \tag{4.1.3}$$

The wavelength is inversely proportional to its frequency and this is denoted as $\lambda = c/v$. The value of Planck's constant h is 6.63×10^{-34}J.s, Boltzmann's constant k is $1.38 \times 10^{-23}\text{JK}^{-1}$ and the speed of light c is 3.0×10^8 ms^{-1}. Ultimately, Planck's assumption of energy quantization and Einstein's photon hypothesis became the fundamental basis for the development of Quantum Mechanics [24].

The Planck's function denoted $B_\lambda(T)$ which varies with wavelength λ and absolute temperature T is given by the equation below.

$$B_\lambda(T) = \frac{2\pi hc^2}{\lambda^5\left(e^{\frac{hc}{\lambda kT}} - 1\right)} \tag{4.1.4}$$

The above Planck function can be written in a more compact form as

$$B_\lambda(T) = \frac{a}{\lambda^5(e^{\frac{b}{\lambda T}} - 1)} \tag{4.1.5}$$

Where $B_\lambda(T)$ is spectral power density, λ is the wavelength in metres, T is the absolute temperature in Kelvins, a=$2\pi hc^2$ and $b = \frac{hc}{k}$ [25].

4.1.2 Creating Blackbody radiation curves

Python computer programs that calculate the blackbody radiant power, plot the curves and analyze data were created as shown in appendix 1 on pages 81-91. The first program generated curves for absolute temperatures $T = 288\ K$ and $T = 5\ 775\ K$, which are shown in Figure 1 and Figure 2, respectively.

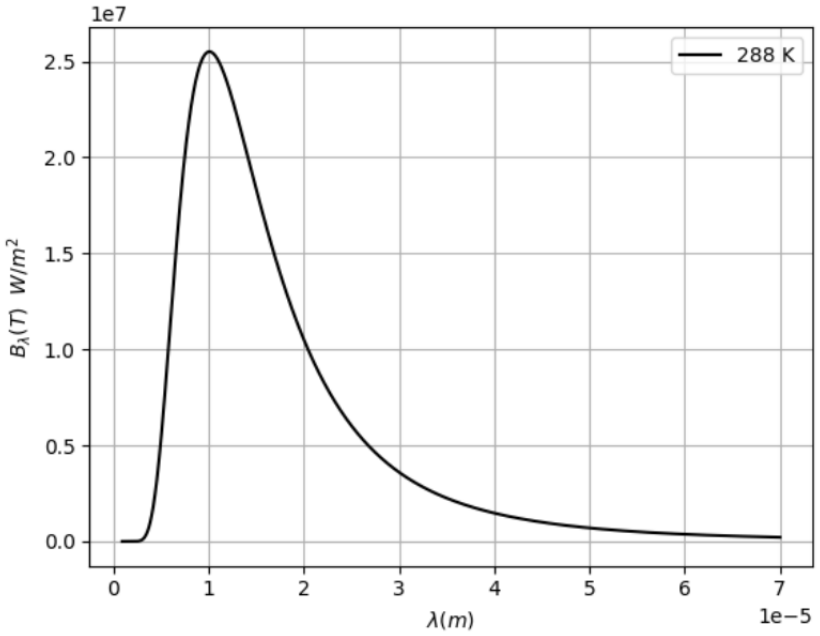


Figure 1: Blackbody radiation Curve at T = 288K

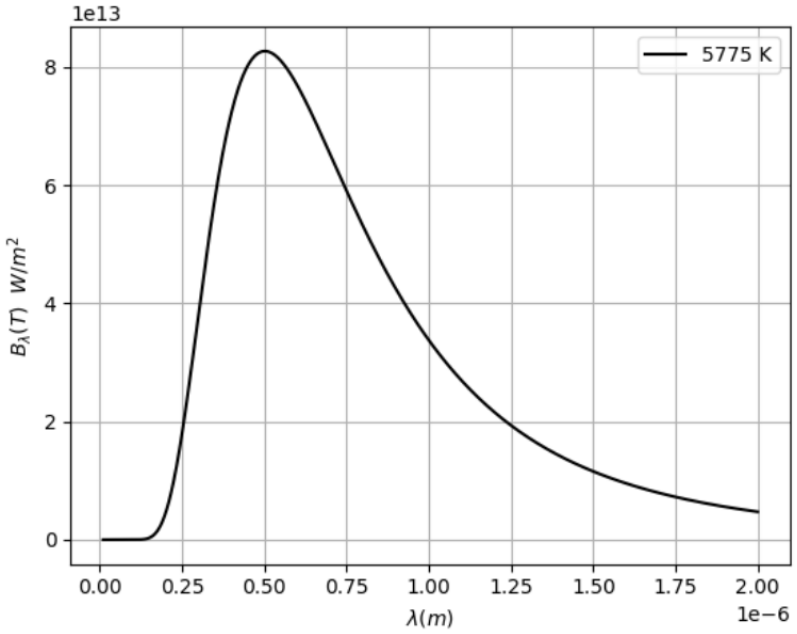


Figure 2: Blackbody radiation curve at T = 5775K

The procedure was repeated for absolute temperatures of 5000K, 7000K, 8000K, 9000K and 10000K. These blackbody radiation curves with the exception of the one at $T = 288K$ (due to different wavelength range) were plotted on the same graph as shown in Figure 3.

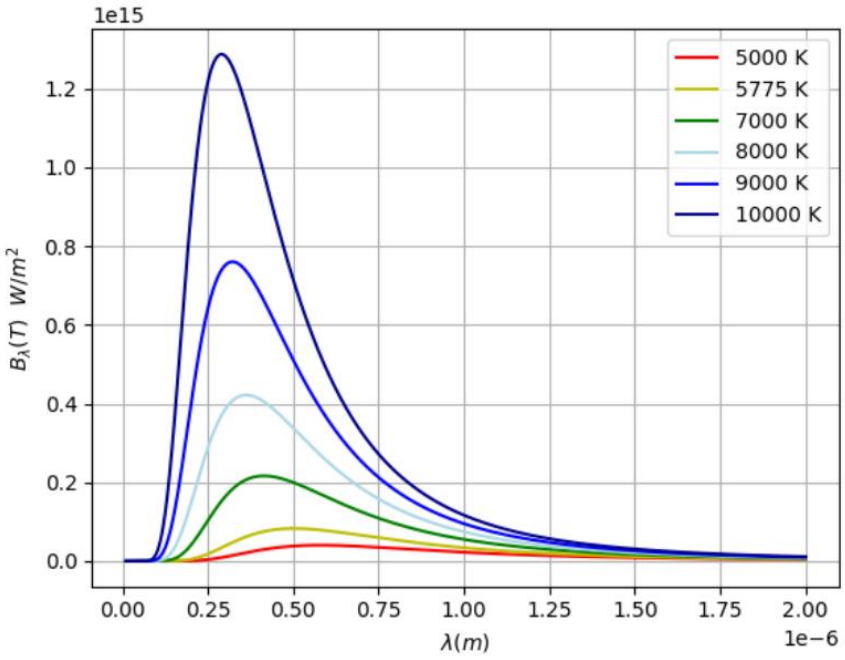


Figure 3:Blackbody radiation curves at specified absolute temperatures

Thereafter, the peak wavelength for each curve (T = 5000K, 5775K, 7000K, 8000K, 9000K, and 10 000K) was found and the results obtained were 5.7939594×10^{-7}, $5.01778178 \times 10^{-7}$, $4.14010401 \times 10^{-7}$, $3.62265227 \times 10^{-7}$, $3.21864186 \times 10^{-7}$, $2.89821982 \times 10^{-7}$, respectively. The peak wavelengths were then plotted against temperature as shown in Figure 4.

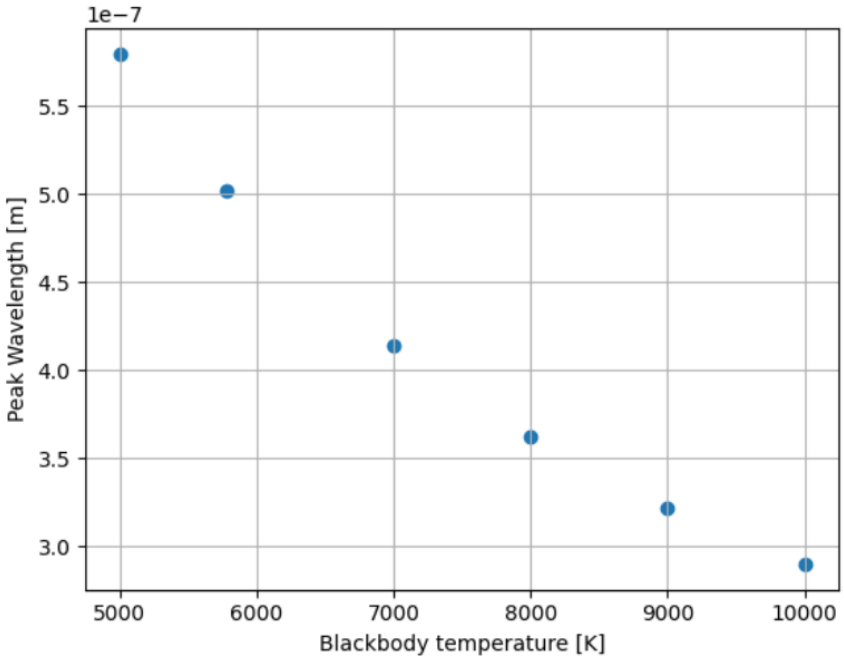


Figure 4: A plot of peak wavelengths at specified temperatures

It can be seen from Figure 4 that the higher the temperature the lower the peak wavelength and the color of the spectrum shifts from reddish to blueish. When the temperature decreases the peak wavelength increases and the color of the spectrum shifts from blueish to reddish.

4.1.3 Using Wien's law to calculate the peak wavelength

According to Wien's Law the maximum wavelength is given by

$$\lambda_{max} = \frac{\alpha}{T} \tag{4.1.6}$$

where; T is the temperature and $\alpha = 2898\ \mu m\text{K}$

Using equation (4.1.6), a computer program was created to obtain the peak wavelength at absolute temperatures $\text{T} = 5000\text{K},\ 5775\text{K}, 7000\text{K}, 8000\text{K}, 9000\text{K},$ and $10\,000\text{K}$. The program yielded the following results: $5.79600000 \times 10^{-7}\text{m}$, $5.01818182 \times 10^{-7}\text{m}$, $4.14000000 \times 10^{-7}\text{m}$, $3.62250000 \times 10^{-7}\text{m}$, $3.22000000 \times 10^{-7}\text{m}$, $2.89800000 \times 10^{-7}\text{m}$.

4.1.4 Comparison of peak wavelengths from Wien's law and Planck's law

A graphical comparison of the peak wavelengths obtained from Wien's law and Planck's law was done as shown in Figure 5 below.

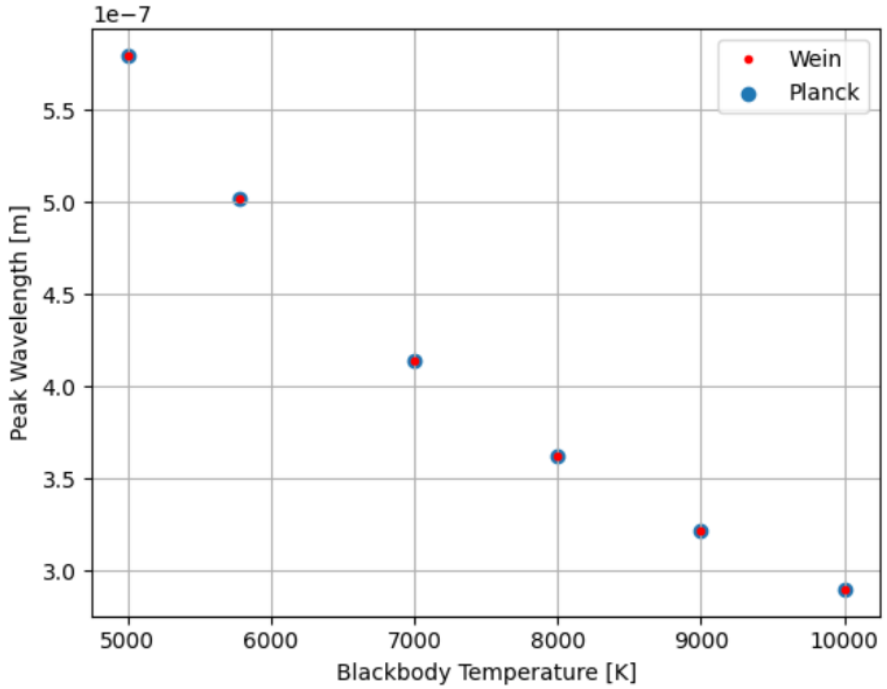


Figure 5: Comparison of peak wavelengths by Planck's law and Wien's law

4.1.5 Root Mean Square Error for peak wavelengths

The Root Mean Square Error (RMSE) measures the average discrepancy deviation between calculated and actual values. It is given by

$$Root\ Mean\ Square\ Error = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}} \tag{4.1.7}$$

where, n is the length of the values, y_i represents the actual values and $\hat{y}_i$ refers to the experimental values [26].

Two computer programs, one, that calculates the root mean square error and the other, that computes the average percentage error for the peak wavelengths were generated. After running the

programs, the root mean square error was found to be $1.020698648873166 \times 10^{-10}$ and the average percentage error obtained was 0.02%.

4.1.7 Calculating the Blackbody radiant power using the Stefan-Boltzmann's law

Stefan-Boltzmann's law is given by:

$$\int_0^\infty B_\lambda(\mathrm{T})d\lambda = \sigma \mathrm{T}^4 \tag{4.1.8}$$

where $\sigma = 5.6696 \times 10^{-8}$

Based on Stefan-Boltzmann's law as presented in equation (4.1.8), a computer program was created to calculate the blackbody radiant power at absolute temperatures T = 5000K, 5775K, 7000K, 8000K, 9000K and 10 000K. The program gave the following results: 35435000.0 W, 63060930.4 W, 136127096.0 W, 232226816.0 W, 371982456.0 W and 566960000.0 W, respectively. The radiant power was plotted against the absolute temperature as shown in Figure 6.

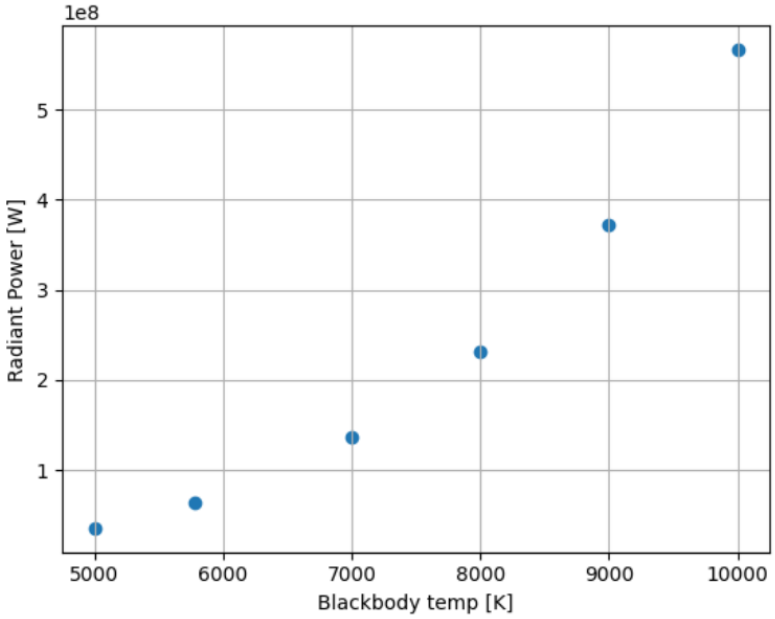


Figure 6: Stefan Blackbody radiant power

4.1.8 Calculating radiant power using Simpson's numerical integration

In this section, calculations for radiant power at absolute temperatures T = 5000K, 5775K, 7000K, 8000K, 9000K and 10 000K were done by using a computer program which employed the Simpson Numerical Integration and utilized the Simpson () function from the scipy.integrate module. This yielded the following results: 32407031.18W, 59275040.64W, 131124610.40W, 226228140.63W, 364994784.57W, 558998266.84W. The blackbody radiant power was then plotted against absolute temperature as shown in Figure 7.

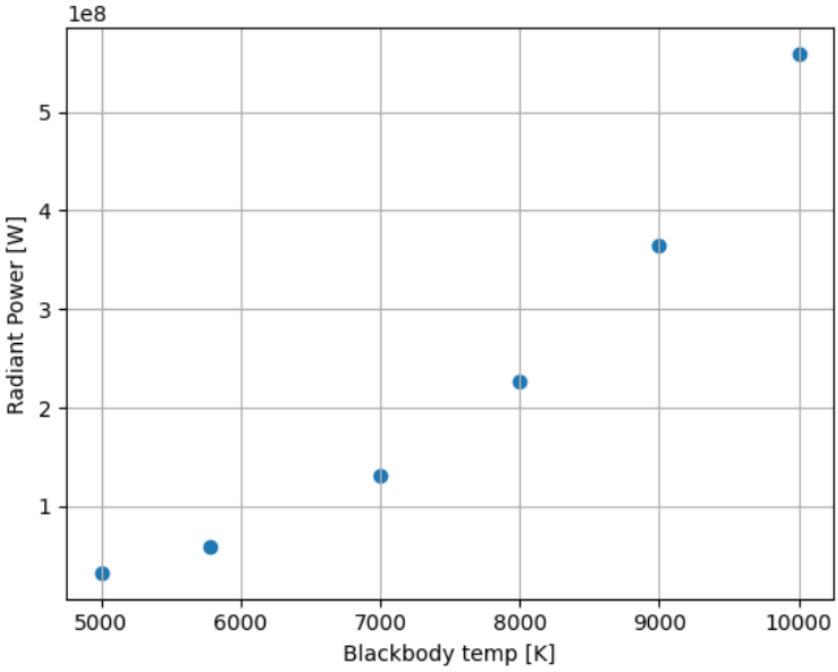


Figure 7: Planck Blackbody Radiant Power

Using the values obtained from section 4.1.7 and section 4.1.8, were plotted on the same axes to make a graphical comparison.

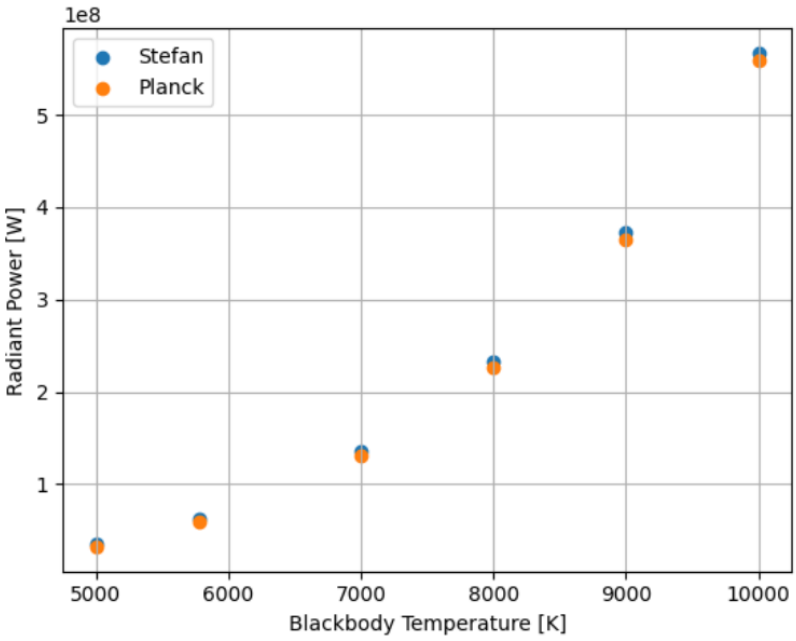


Figure 8: Comparison of Blackbody radiant power from Stefan's law with Planck's law

4.1.8.1 Calculating the average percentage error for the radiant power

The average percentage error obtained by using Simpson's rule was then calculated. The percentage error defines the accuracy of a measurement compared to a known or accepted value. It is given by the following equation

$$Percentage\ Error = \left|\frac{Experimental\ Value - Actual\ Value}{Actual\ Value}\right| x\ 100\% \qquad (4.1.9)$$

Equation (4.1.9) was then transformed into a computer program which found the average percentage error to be 4.26%.

4.1.9 Blackbody Interactive Plot

Then an interactive plot was created by integrating Ipywidgets into the computer program that plots blackbody radiation curves. The process and functionality of the applet developed are explained in figure 9 shown below, providing a visual depiction of its outlook and operational principles.

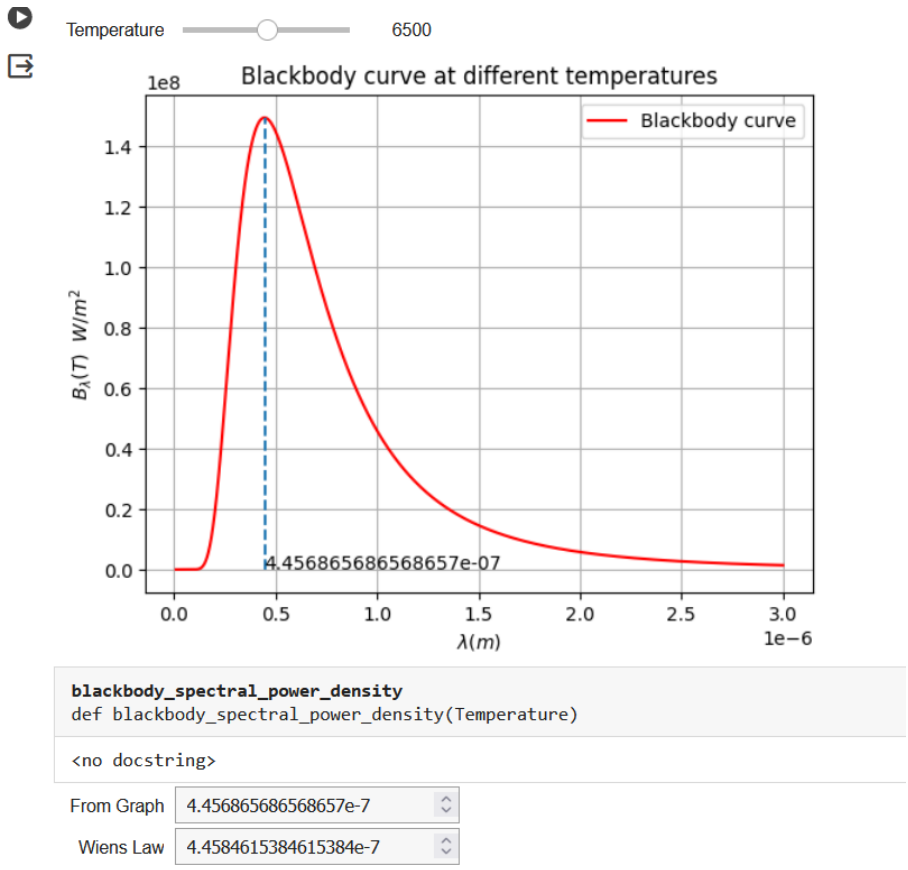


Figure 9: Blackbody radiation curve applet

The blackbody applet shown in figure 9 plots the blackbody curves for varying absolute temperatures using a slider, and annotates the peak wavelength. It also gives a comparison between peak wavelength values at each temperature obtained from the graph and those calculated using Wien's law. The user uses the slider to change the temperature input and observes the changes in the graph as well as changes in the peak wavelengths from the graph and Wien's law. As can be seen from figure 9, at temperature 6500K the value for the wavelength taken from the graph is 4.57×10^{-7}m while the calculated value from Wien's law gives 4.58×10^{-7}m this can help users to discern that Wien's law provides an accurate approximation.

For easy distribution and access, the applet was converted into an executable file so that users can install on their computers and run the program without using the code editor and the source code.

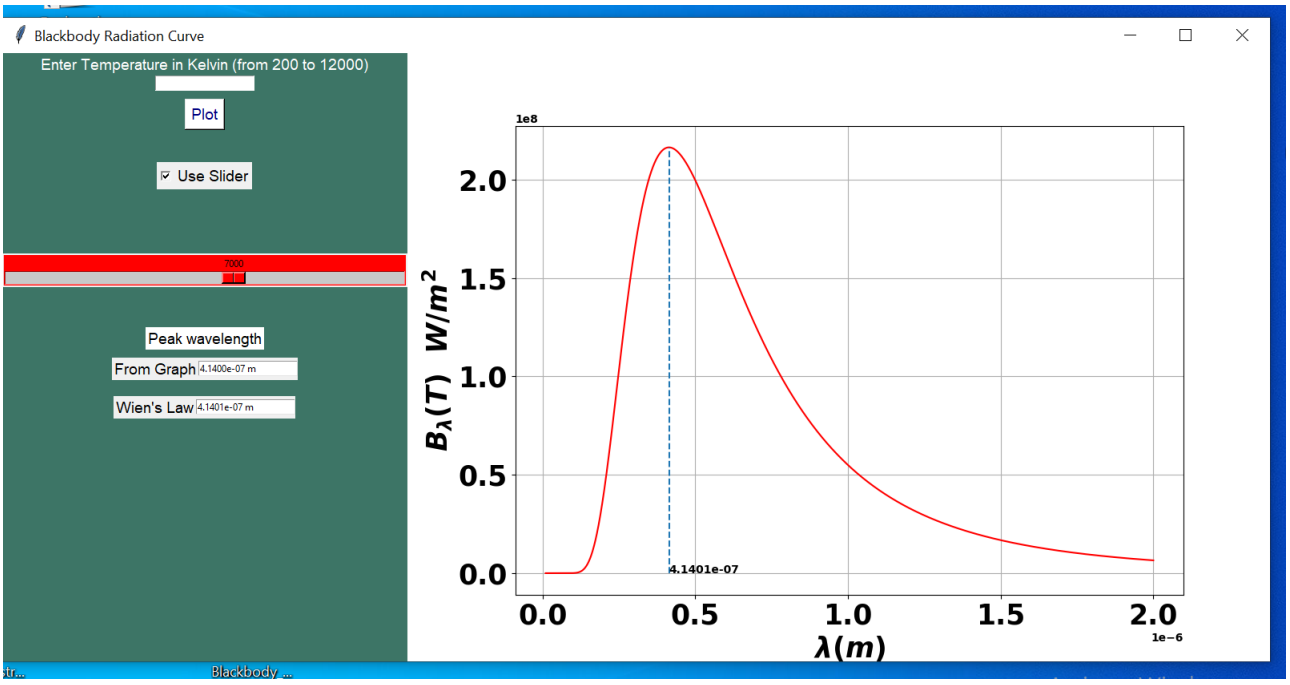


Figure 10: Blackbody applet installed on windows

4.1.10 Discussion

The blackbody radiation curves were plotted using appropriate ranges of wavelengths that were selected based on the absolute temperatures. For absolute temperatures less or equal to 2 000 K wavelengths ranged from 900nm to 0.7μm while for absolute temperatures above 2 000K the range was taken from 0.01μm to 2μm. The blackbody curve with absolute temperature, T=288K could not be plotted on the same graph with the ones with higher temperatures due to different wavelength ranges.

The maximum wavelength of each curve was obtained and compared to the value calculated from Wien's law. The root mean square error obtained was $1.020698648873166 \times 10^{-10}$. The calculated average percentage error was found to be 0.02%.

To obtain the total power density, the area under each curve was calculated using the Simpson function from the scipy.integrate module. The values obtained were compared to the ones calculated using the Stefan-Bolzmann's law. The average percentage error was found to be 4.26%.

The applet provides a user interface comprising an entry box for temperature input and a slider enabling dynamic temperature adjustments. Upon inputting a temperature value, the corresponding blackbody curve is plotted, allowing users to visualize how the temperature affects the radiant power density and the peak wavelength. The applet identifies the maximum point on the curve and annotates its wavelength value, aiding users in understanding how an increase or decrease in temperature affects peak wavelength value. The peak wavelength value obtained from the graph is compared to a value calculated using Wien's law to show how the two relates. The slider functionality further enhances interactivity, enabling users to explore how changes in the temperature influence the blackbody curve in real-time.

4.1.11 Conclusion

The applet created offers a user-friendly platform for interactive exploration of mathematical concepts focusing on Planck's blackbody radiation law as well as Wien's law. Users are able to interact with the applet and get results in real-time. This makes it possible for the user to gain a deeper understanding of how the change in temperature affects the radiant power of a blackbody and its peak wavelength within the shortest period of time.

4.2 Blackbody Energy Fractions

4.2.1 Theory

Blackbody fractions refer to the proportion of radiation emitted by a blackbody within a specific wavelength range or band. Blackbody fractions are an integral part in calculating emissive power of a given surface at any given temperature in a specified range of wavelengths. The spectral radiance distribution for a blackbody is provided by the Planck's radiation law.

The total power emitted by the surface is given by the integral

$$F = \int_0^\infty E_\lambda B_\lambda(T) d\lambda \tag{4.2.1}$$

In the equation, F represents the total power or total radiant flux, E_λ is the spectral emissivity and, $B_\lambda(T)$ corresponds to the Planck's function. For a blackbody the spectral emissivity (E_λ) is equal to 1 [25].

Hence,

$$F = \int_0^\infty B_\lambda(T) d\lambda \tag{4.2.2}$$

The expression for Planck's function is:

$$B_\lambda(T) = \frac{a}{\lambda^5(e^{\frac{b}{\lambda T}} - 1)} \tag{4.2.3}$$

In equation (4.2.3), let $a = 2\pi hc^2$ and $b = \frac{hc}{k}$ where, h is the Planck's constant, c is the speed of light in a vacuum and k is the Boltzmann's constant.

By substituting equation (4.2.3) into equation (4.2.2), we obtain:

$$F = \int_0^\infty \frac{a}{\lambda^5(e^{\frac{b}{\lambda T}} - 1)} d\lambda \tag{4.2.4}$$

But we know that this integral gives the Stefan-Bolztmann's law

$$F = \int_0^\infty \frac{a}{\lambda^5(e^{\frac{b}{\lambda T}} - 1)} d\lambda = \sigma T^4 \tag{4.2.5}$$

where $\sigma = \frac{2C_1\pi^5}{15\,C_2^4}$, $C_1 = hc^2$ and $C_2 = \frac{hc}{k}$ [27]

The fraction of energy transmitted by the wavelength can be determined by considering wavelengths in the range from 0 to λ.

$$f_\lambda(T) = \frac{\int_0^{\lambda T} B_\lambda(T)d_\lambda}{\int_0^{\infty} B_\lambda(T)d_\lambda} = \frac{\int_0^{\lambda T} \frac{a}{\lambda^5\left(e^{\frac{b}{\lambda T}} - 1\right)} d\lambda}{\int_0^{\infty} \frac{a}{\lambda^5\left(e^{\frac{b}{\lambda T}} - 1\right)} d\lambda} \tag{4.2.6}$$

From equation (4.2.5) we know that

$$\int_0^{\infty} \frac{a}{\lambda^5\left(e^{\frac{b}{\lambda T}} - 1\right)} d\lambda = \sigma T^4$$

$$\therefore f_\lambda(T) = \frac{1}{\sigma T^4}\int_0^{\lambda T} \frac{a}{\lambda^5\left(e^{\frac{b}{\lambda T}} - 1\right)} d\lambda \tag{4.2.7}$$

We now define $x = \lambda T$, which implies $\lambda = \frac{x}{T}$. Therefore, $d\lambda = \frac{dx}{T}$

By substituting $d\lambda = \frac{dx}{T}$ into equation (4.2.6), we get

$$f(\lambda T) = f(x) = \frac{1}{\sigma T^4}\int_0^{x} \frac{\frac{a}{T}dx}{\left(\frac{x}{T}\right)^5\left(e^{\frac{b}{x}} - 1\right)} \tag{4.2.8}$$

This reduces to

$$f(x) = \frac{a}{\sigma}\int_0^{x} \frac{dx}{x^5\left(e^{\frac{b}{x}} - 1\right)} \tag{4.2.9}$$

Equation (4.2.9) cannot be integrated directly as it blows up to infinite for $x^5\left(e^{\frac{b}{x}} - 1\right) = 0$.

We now set $w = \frac{b}{x}$, therefore $x = \frac{b}{w}$ or $x = bw^{-1}$ hence $dx = -2\frac{b}{w^2}dw$

We substitute $x = \frac{b}{w}$ and $dx = -2\frac{b}{w^2}dw$ into equation (4.2.9)

We now get the limits of integration since we have $w = \frac{b}{x}$, when $x = 0$, $w \rightarrow \infty$ for the lower limit and $x = x$, $w = \frac{b}{x}$

The integral now becomes

$$f(x) = \frac{a}{\sigma}\int_{\infty}^{\frac{b}{x}} \frac{\frac{-b}{w^2}}{\left(\frac{b}{w}\right)^5 (e^w - 1)} dw$$

$$= \frac{a}{b^4\sigma}\int_{\infty}^{\frac{b}{x}} \frac{\frac{-w^5}{w^2}}{e^w - 1} dw$$

$$\therefore f(x) = -\frac{a}{b^4\sigma}\int_{\infty}^{\frac{b}{x}} \frac{w^3}{e^w - 1}\, dw \qquad (4.2.10)$$

Dividing both the numerator and the denominator by e^w we get

$$f(x) = -\frac{a}{b^4\sigma}\int_{\infty}^{\frac{b}{x}} \frac{w^3 e^{-w}}{1 - e^{-w}} dw \qquad (4.2.11)$$

Equation (4.2.11) can be re-written as

$$f(x) = -\frac{a}{b^4\sigma}\int_{\infty}^{\frac{b}{x}} w^3 e^{-w}\left(\frac{1}{1 - e^{-w}}\right) dw \qquad (4.2.12)$$

But from the binomial expansion we know that,

$$\frac{1}{1-z} = 1 + z + z^2 + z^3 + z^4 + z^5 \ldots\ldots + z^n \qquad [28]$$

Therefore $\frac{1}{1- e^{-w}} = 1 + e^{-w} + e^{-2w} + e^{-3w} + e^{-4w} + e^{-5} + e^{-nw}$

Using this in equation (4.2.12), we, write it as

$$f(x) = -\frac{a}{b^4\sigma}\int_{\infty}^{\frac{b}{x}} x^3 e^{-w}(1 + e^{-w} + e^{-2w} + e^{-3w} + e^{-4} + e^{-5} + \ldots + e^{-nw})dw \qquad (4.2.13)$$

This gives us

$$f(x) = \frac{-a}{b^4\sigma}\int_{\infty}^{\frac{b}{x}} w^3 e^{-nw}\, dw \qquad (4.2.14)$$

Let us now integrate $\int_{\infty}^{\frac{b}{x}} w^3 e^{-nw} dw$ by parts

We let $u = w^3$ and $dv = e^{-nw}dw$. Therefore, $du = 3w^2 dw$

and

$$v = \int_{\infty}^{\frac{b}{x}} e^{-nw} dw$$

$$v = \frac{-1}{n} e^{-nw}$$

We now apply

$$\int udv = uv - \int vdu$$

$$\int_{\infty}^{\frac{b}{x}} we^{-nw} dw = w^3\left(-\frac{e^{-nw}}{n}\right) - \int\left(-\frac{e^{-nw}}{n}\right) 3w^2 dw \qquad (4.2.15)$$

This gives

$$\int_{\infty}^{\frac{b}{x}} w^3 e^{-nw} dw = -\frac{e^{-nw}w^3}{n} + \frac{1}{n}\int 3w^2 e^{-nw} dw \ldots (4.2.16)$$

We let

$$-\frac{e^{-nw}w^3}{n} = I_1$$

and we work out

$$\frac{1}{n}\int 3w^2 e^{-nw} dw$$

We now let $u = 3w^2$ and $dv = e^{-nw}dw$

This gives us $du = 6wdw$

and

$$v = \int e^{-nw} dw$$

$$\therefore \ v = -\frac{e^{-nw}}{n}$$

We apply

$$\int udv = uv - \int vdu$$

$$\frac{1}{n}\int e^{-nw} 3w^2 dw = \frac{1}{n}\left[-3w\frac{e^{-nw}}{n} - \int_{\infty}^{\frac{b}{x}} -\frac{e^{-nw}}{n} 6w dw\right] \tag{4.2.17}$$

$$\frac{1}{n}\int e^{-nw} 3w^2 dw = -3w^2\frac{e^{-nw}}{n^2} + \frac{1}{n^2}\int_{\infty}^{\frac{b}{x}} 6w^{-nw} dw \tag{4.2.18}$$

We let

$$-3w^2\frac{e^{-nw}}{n^2} = I_2$$

We now solve

$$\frac{1}{n^2}\int_{\infty}^{\frac{b}{x}} 6w^{-nw} dw$$

Letting $u = 6w$ and $dv = e^{-nw}$ we get $du = 6wdw$

and

$$dv = \int_{\infty}^{\frac{b}{x}} e^{-nw} dw$$

$$\therefore v = \frac{e^{-nw}}{n}$$

applying

$$\int udv = uv - \int vdu$$

we get

$$\frac{1}{n^2}\int_{\infty}^{\frac{b}{x}} 6w^{-nw}dw = \frac{1}{n^2}\left[6w\left(-\frac{e^{-nw}}{n}\right) - \int_{\infty}^{\frac{b}{x}} -\frac{e^{-nw}}{n}6dw\right]$$

$$= \frac{1}{n^2}\left[-\frac{6we^{-nw}}{n} + \frac{6}{n^2}\int_{\infty}^{\frac{b}{x}} e^{-nw}dw\right]$$

$$= \frac{1}{n^2}\left[-\frac{6we^{-nw}}{n} - \frac{6e^{-nw}}{n^2}\right]$$

$$= -\frac{6we^{-nw}}{n^3} - \frac{6e^{-nw}}{n^4} \ldots (4.2.19)$$

We let

$$-\frac{6we^{-nw}}{n^3} - \frac{6e^{-nw}}{n^4} = I_3$$

From equation (4.2.14)

$$f(x) = -\frac{a}{b^4\sigma}\int_{\infty}^{\frac{b}{x}} w^3e^{-nw}dw$$

$$f(x) = -\frac{a}{b^4\sigma}(I_1 + I_2 + I_3)$$

$$f(x) = -\frac{a}{b^4\sigma}\left(-\frac{w^3e^{-nw}}{n} - \frac{3w^2e^{-nw}}{n^2} - \frac{6we^{-nw}}{n^3} - \frac{6e^{-nw}}{n^4}\right) \quad (4.2.20)$$

$$f(x) = \frac{a}{b^4\sigma}\left(\frac{e^{-nw}}{n}\right)\left(w^3 + \frac{3w^2}{n} + \frac{6w}{n^2} + \frac{6}{n^3}\right)$$

$$f(x) = \frac{a}{b^4\sigma}\sum_{n=1}^{\infty}\frac{e^{-nw}}{n}\left(w^3 + \frac{3w^2}{n} + \frac{6w}{n^2} + \frac{6}{n^3}\right) \quad (4.2.21)$$

But we know that $a = 2\pi hc^2$, $b = \frac{hc}{k}$ and $\sigma = \frac{2C_1\pi^5}{15\,C_2^4}$ where $C_1 = hc^2$ and $C_2 = \frac{hc}{k}$

then

$$\frac{a}{b^4\sigma} = \frac{2\pi hc^2}{\left(\frac{hc}{k}\right)^4\frac{2C_1\pi^5}{15{C_2}^4}} = \frac{2\pi h^5c^6k^4.15}{2\pi^5h^5c^6k^4} = \frac{15}{\pi^4}$$

and

$$f(x) = \frac{15}{\pi^4}\sum_{n=1}^{\infty}\frac{e^{-nw}}{n}\left(w^3 + \frac{3w^2}{n} + \frac{6w}{n^2} + \frac{6}{n^3}\right) \tag{4.2.22}$$

replacing w with $\frac{b}{x}$ we get

$$f(x) = \frac{15}{\pi^4}\sum_{n=1}^{\infty}\frac{e^{-n\frac{b}{x}}}{n}\left(\left(\frac{b}{x}\right)^3 + \frac{3\left(\frac{b}{x}\right)^2}{n} + \frac{6\left(\frac{b}{x}\right)}{n^2} + \frac{6}{n^3}\right)$$

$$f(x) = \frac{15}{\pi^4}\sum_{n=1}^{\infty}\frac{e^{-n\frac{b}{x}}}{n}\left(\frac{b^3}{x^3} + \frac{3b^2}{nx^2} + \frac{6b}{n^2x} + \frac{6}{n^3}\right)$$

or

$$f(x) = \frac{15}{\pi^4}\sum_{n=1}^{\infty}e^{-\frac{nb}{x}}\left(\frac{b^3}{nx^3} + \frac{3b^2}{n^2x^2} + \frac{6b}{n^3x} + \frac{6}{4}\right) \tag{4.2.23}$$

Equation (4.2.23) gives us the summation needed to obtain the estimated energy fractions distributed by a blackbody at a particular temperature.

The values of n were set starting from $n = 1$ to $n = 20$. The greater the n end point the more accurate the result will be.

$$f(x) = \frac{15}{\pi^4}\sum_{n=1}^{n=20}e^{-\frac{nb}{x}}\left(\frac{b^3}{nx^3} + \frac{3b^2}{n^2x^2} + \frac{6b}{n^3x} + \frac{6}{4}\right) \tag{4.2.24}$$

After carrying out the derivation, the final equation, equation (4.2.24) was translated into a computer code (appendix 2 pages 92-94) so that it can be solved iteratively. The results were generated inform of a table and a graph. As can be observed from the table in appendix 2 on pages 95-105, the wavelength multiplied by temperature represented by x ranged from 1100μm to 50000μm with a step size of 100μm. The results gave the blackbody fraction depicted by f(x). Thereafter, a graph of the calculated blackbody fractions against wavelength multiplied by temperature was plotted as shown in Figure 11. Then a comparative analysis between the values calculated using equation (4.2.24) and those obtained from Sol Wieder's "An Introduction to Solar

Energy for Engineers and Scientists" (Wieder, 1982, p.14) as shown in the table in appendix 2 on pages 101-102 was made [25]. The values were plotted on the same graph as shown in Figure 13.

Figure 11 below shows the graph of blackbody fractions calculated using equation (4.2.24).

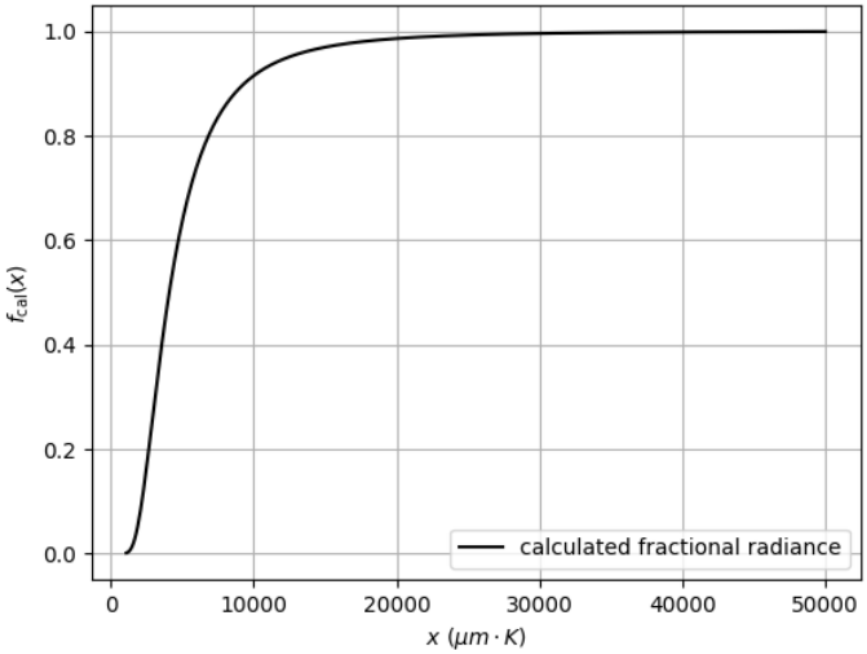


Figure 11: Calculated Blackbody energy fractions

Figure 12 below shows the graph of blackbody energy fractions using the data taken from Sol Wielder (1982) *[25]*.

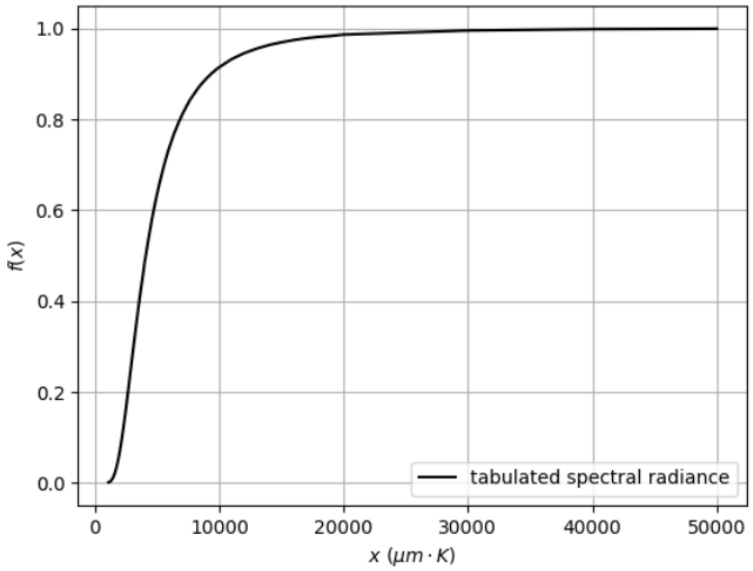


Figure 12: Blackbody energy fractions plot using values from Wieder (Wieder, 1982)

4.2.4 Comparison of calculated and tabulated Blackbody energy fractions

In order to make a visual analysis of the of the blackbody fractions calculated from equation (4.2.24) and those obtained from Sol Wieder, the two graphs were plotted on the same axes.

Figure 13 shows the correlation between the blackbody fractions obtained from Sol Wieder(1982) and calculated values from equation (4.2.24).

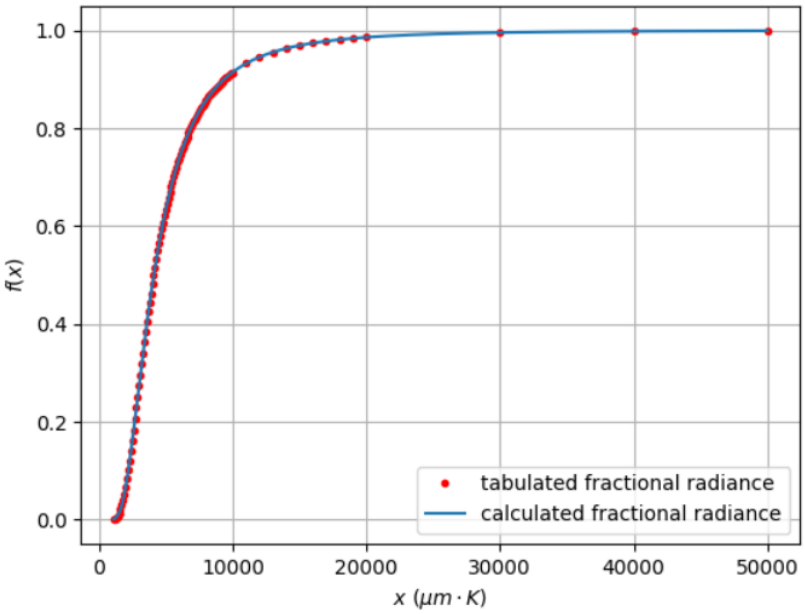


Figure 13: Comparison of tabulated and calculated Blackbody energy fractions

4.2.4.1 Calculating the Mean Root Square Error

In order to numerically assess the accuracy of the calculated values to the values obtained from Sol Wieder the mean root square error (RMSE) was calculated and found to be 0.00036. This is a very low-level error. It means that the calculated values of the fractional radiance differ from the values obtained from the book by approximately 0. 00036μm.K

4.2.5 Blackbody Fractions Applet

Based on equation (4.2.24), an interactive applet for finding blackbody energy fractions between two given wavelengths at specified temperatures was created using Ipywidgets as shown in Figure 14.

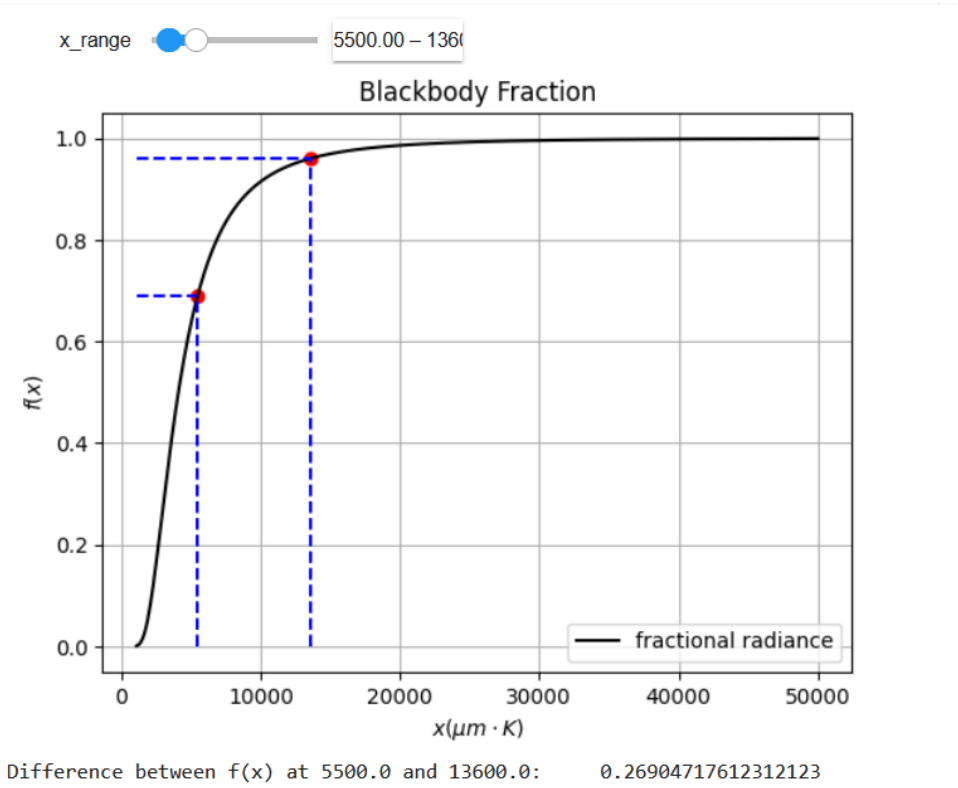


Figure 14: Blackbody Energy Fraction applet

This applet was done in Jupyter Notebook environment and it works in such a way that the user inputs their desired lower and upper limits of the wavelength x temperature values to find the blackbody fraction using the range slider. As can be observed, the lower limit was at 5500μmK with the upper limit at 13600μmK this gave 0.269 which is 26.9%.

The applet was further developed into a fully-fledged application which was installed on windows computer. Figure 15 shows the operation of the fully fledged applet. The blackbody fraction can be determined by entering the values of the lower and upper bounds of wavelength multiplied by temperature into their respective fields. Then, the visual analysis and the blackbody fraction is given out.

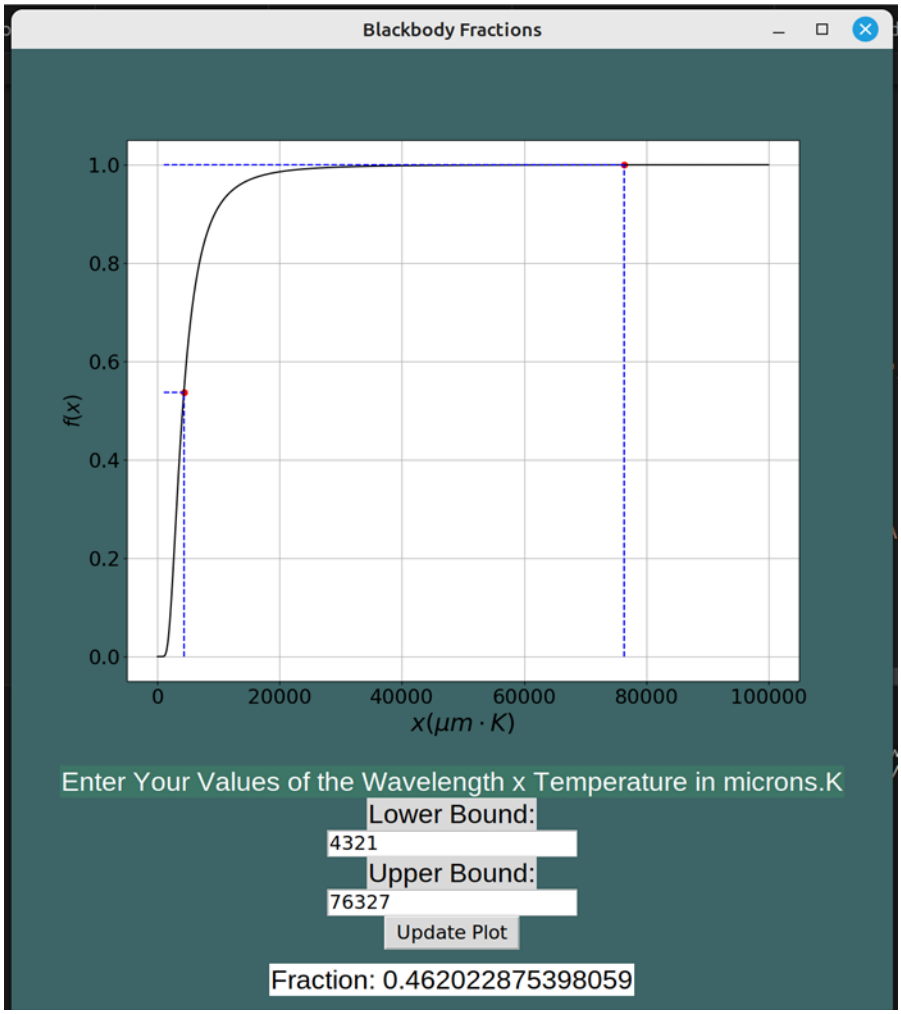


Figure 15: Installed Blackbody Fraction applet

4.2.6 Discussion

The Planck's function was integrated and reduced to an infinite series. The obtained infinite series was used to calculate the blackbody fraction of the energy between two wavelengths at any given temperature. The graph shows that there was no significant difference between the values calculated and the values obtained. The RMSE was found to be 0.00036. This entails a very low level of error. It means that the calculated values of the blackbody fractions differ from the values obtained from Wieder (Wieder, 1982, p. 14) by approximately 0.00036 μmK.

4.2.7 Conclusion

The applet offers a quick and accurate method for calculating and visualizing blackbody energy fractions for specified temperature and wavelength ranges. It is useful in imaging science for analyzing thermal radiation distribution, interpreting temperature variations in images, and combining images from different spectral bands. Additionally, the applet enhances remote sensing data evaluation and interpretation.

4.3 Seasonal Variability of the Eccentricity Correction Factor

4.3.1 Theory

Eccentricity correction factor refers to the term used to accommodate the elliptical form of the Earth's orbit around the Sun. The Earth moves around the Sun in a slightly oval-shaped path, not a perfect circle. This path makes the distance between the Earth and the Sun change throughout the year. When the distance changes, the amount of solar energy that reaches the Earth's atmosphere changes too. If the Earth is farther away from the Sun, it gets less solar energy, and if it's closer, it gets more.

Treating the motion of the earth as a one-body problem and using the Newtonian dynamics, one can show that; the earth moves in a fixed plane round the sun and that its orbit is an ellipse with the sun situated at one of the foci.

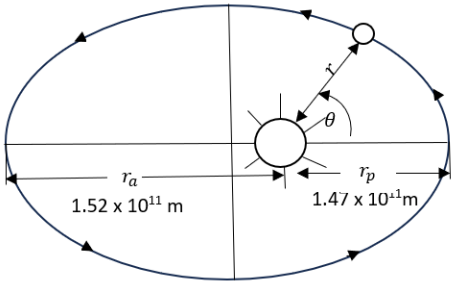


Figure 16: Schematic diagram for Earth's eccentricity correction factor

The earth's orbit is best expressed in polar coordinates as

$$r = \frac{a(1-\epsilon^2)}{1+\epsilon.cos\theta} \tag{4.3.1}$$

where, a, the semi-major axis = 1.479 x 10^{11}m

ϵ, the eccentricity = 0.0167 (a measure of the deviation from the circle) where $\epsilon = \frac{c}{a}$, c being the distance of the focus from the origin. Smallest value of ϵ occurs when $\theta = 0$. This is called perihelion represented by r_p in figure 15

$$r_p = a(1-\epsilon) = 1.497 \times 10^{11}(1-0.0167)$$

$$= 1.471 \times 10^{11}\text{m}$$

Maximum value of r occur when $\theta = 180°$. This is called aphelion r_a

$$r_a = a(1+\epsilon) = 1.497 \times 10^{11}(1+0.0167) = 1.521 \times 10^{11}\text{m}$$

Perihelion and aphelion distances differ by less than 2% from the mean value. Therefore, the earth's orbit is almost a perfect circle. Aphelion usually occurs on 4th July while perihelion happens to 4th January. [25]

Spencer derived a detailed expression for calculating the E_0 for the Earth's orbits around the Sun given by:

$$E_0 = \left(\frac{r_0}{r_2}\right)^2 \tag{4.3.2}$$

$$E_0 = 1.00110 + 0.034221\cos\Gamma + 0.00128\cos\Gamma + 0.000719\cos 2\Gamma + 0.000077\sin 2\Gamma \tag{4.3.3}$$

where, r_0 is the mean Earth - Sun distance in AU, r_2 is the Earth - Sun distance in AU on a specific day of 365 days and Γ is the day angle measured in radians.

The day angle refers to the angle used to monitor the position of the Earth with respect to the Sun for any particular day of the year. The day angle helps find where the sun is in the sky, making it easier to position solar panels for maximizing energy capture [29]. It is given by the following equation:

$$\Gamma = 2\pi\left(\frac{d_n - 1}{365}\right) \tag{4.3.4}$$

d_n is the day number, e.g. d_n = 1 for January 1, $d_n = 30$ for January 30, $d_n = 35$ for February 4.

Duffie and Beckmann also developed an elementary equation which can be used to calculate eccentricity correction factor.

$$E_0 = 1.033cos\left(\frac{2\pi d_n}{365}\right) \tag{4.3.5}$$

[12]

4.3.2 Using data from "An Introduction to Solar Radiation" [12]

A Python file was created using the Gedit text editor. This file contained a table of values of eccentricity correction factor, as shown by Iqbal (1983. p. 4,5). This file was named ecf_data and saved in the same location as the code. Then a python program was developed to generate the graph and the table of eccentricity correction factor for each day of the year.

The output of the computer program is shown in Figure 17 and tabulated data in appendix 3 on pages 103-105, depicting the eccentricity correction factor for each day of the year. Please note that in the tables in appendices containing months, February is taken to have 28 days, so there is no variable assigned to day number 29, 30 and 31 for this month. For April, June, September and November no variable is assigned to day number 31 since the last day for these months is 30^{th}.

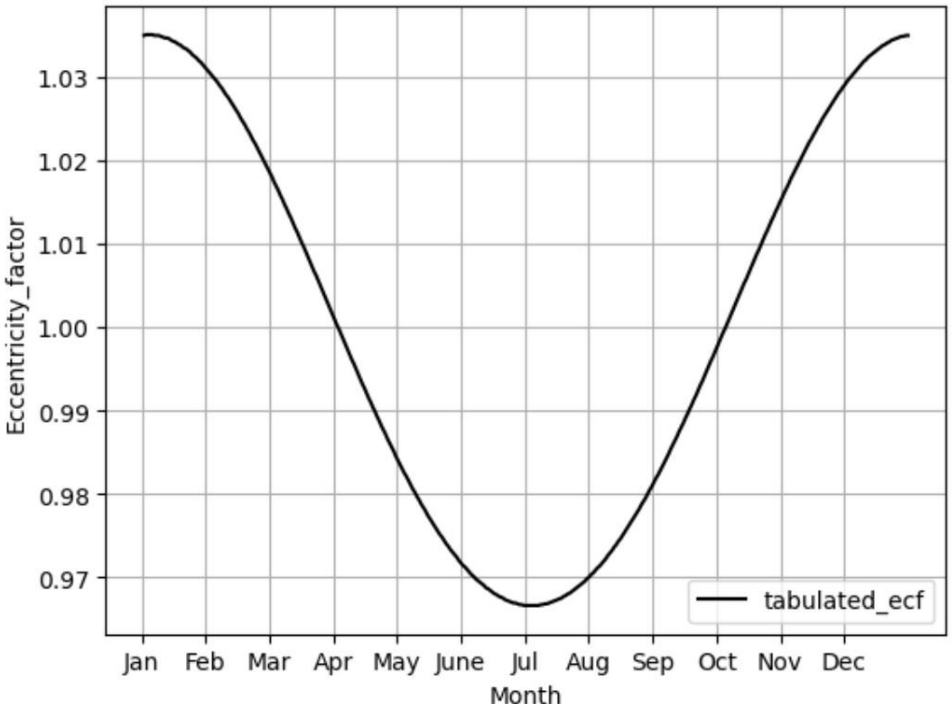


Figure 17: Plot of Eccentricity Correction Factor

4.3.3 Calculating Eccentricity Correction Factor using Spencer's expression

As noted in section 4.3.1 Spencer came up with an expression of calculating eccentricity correction factor. Using the same expression shown in equation (4.3.3), a computer code (appendix 3 pages 106-109) to calculate and generate graphical and tabular data of eccentricity correction factor for each day of the year was generated. The code generated the graph of eccentricity correction factor as shown in Figure 18 and a table shown in appendix 3 on pages 110-112.

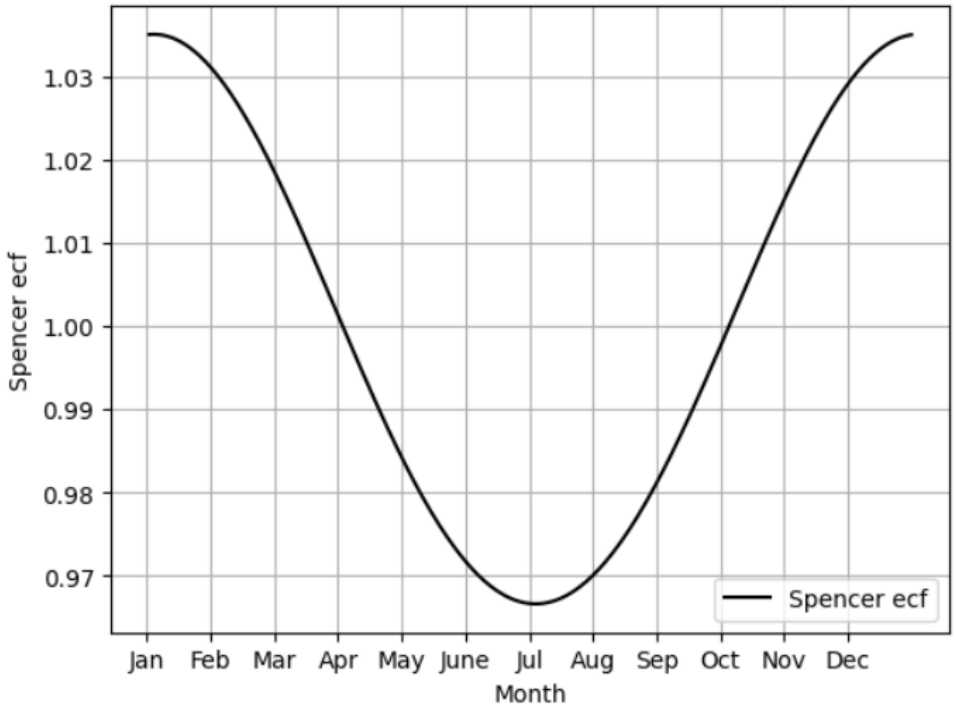


Figure 18: Eccentricity Correction Factor using Spencer's expression

4.3.3.1 Root Mean Square Error from Spencer's expression

The root mean square error of eccentricity correction factor from Spencer's expression and those obtained from "An Introduction to Solar Radiation" [12] was calculated to be 4.91×10^{-5} while the percentage error was found to be 0.00%.

4.3.4 Duffie and Beckmann expression

Next, using the Duffie and Beckmann's expression which is shown in equation (4.3.5), a computer code (appendix 3 pages 113-117) was created to compute the eccentricity correction factor. The output of the code is shown in Figure 19 and a table on pages 118-120.

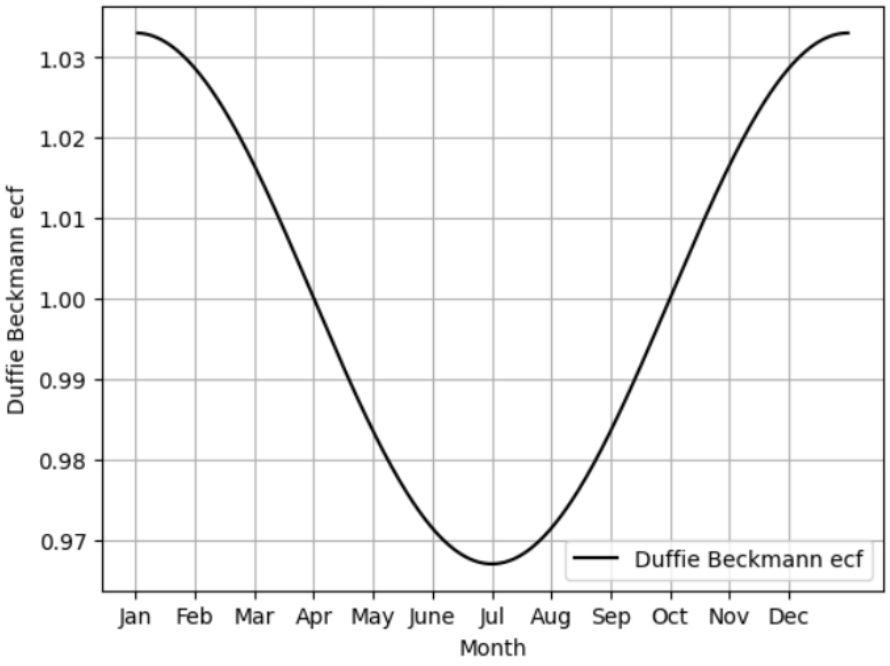


Figure 19: Plot of Eccentricity Correction Factor from Duffie and Beckmann's expression

4.3.4.1 Root Mean Square Error from Duffie and Beckmann's expression

After computing and plotting the values of eccentricity correction factor from Duffie and Beckmann's equation, the accuracy of these values compared to those obtained from "An Introduction to Solar Radiation" [12] was calculated inform of the root mean square error. The root mean square error was found to be 0.00168 and the average percentage error was calculated to be 0.14%.

4.3.5 Graphical comparison

A graphical comparison of the values obtained from equations (4.3.3) and (4.3.5) and those from "An Introduction to Solar Radiation" [12] was then done as shown in Figure 20.

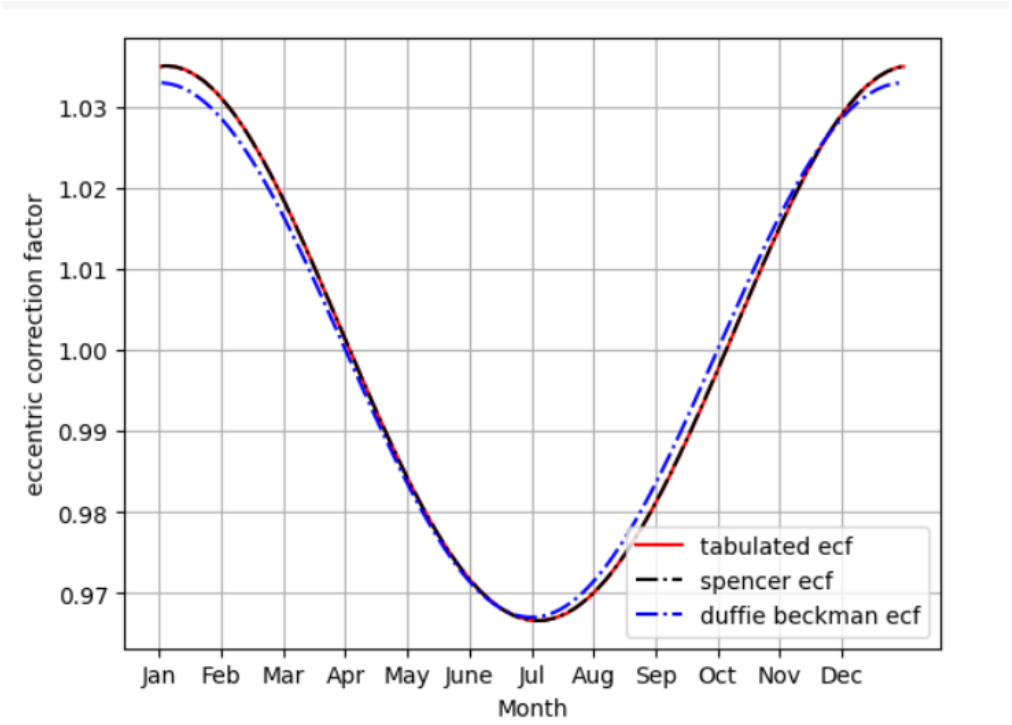


Figure 20: Spencer's, Duffie and Beckman's expressions and values from Iqbal (1983)

4.3.6 Creating Interactive Graphs

In order to create interactivity and add a graphical user interface a computer code was developed using Ipywidgets which yielded the applet shown in Figure 21.

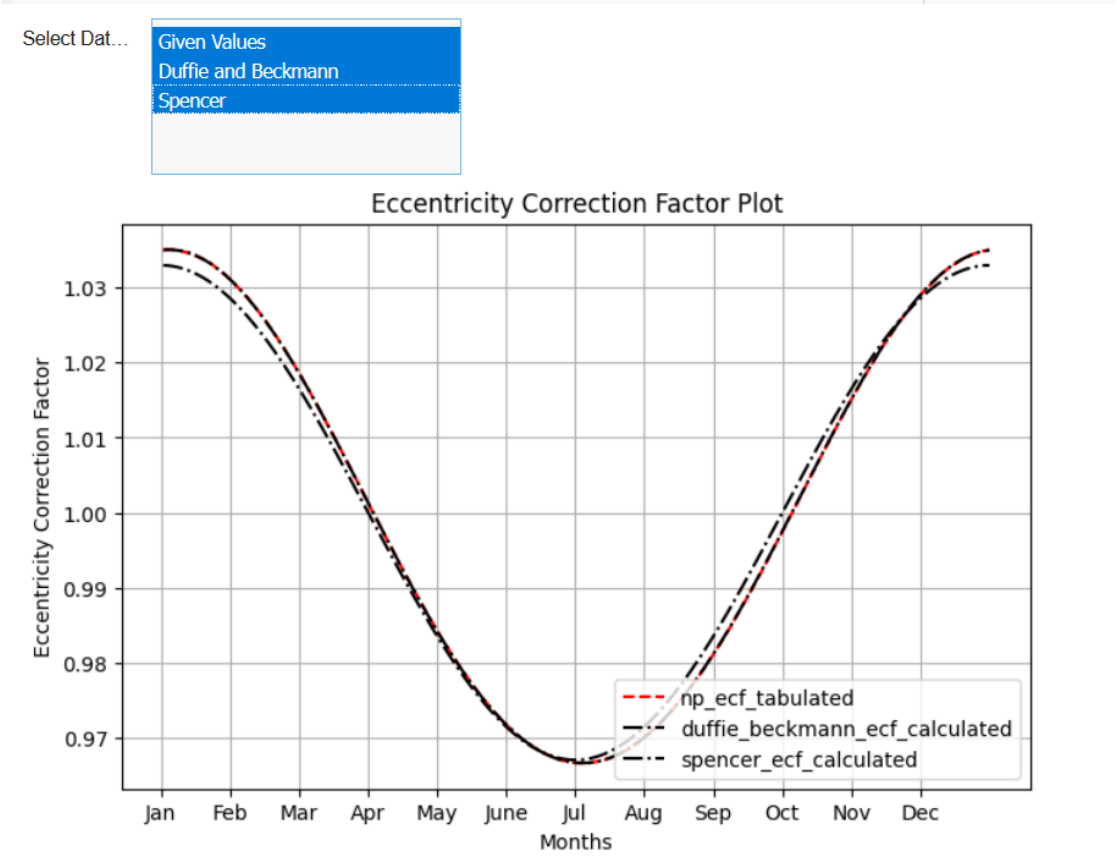


Figure 21: Eccentricity Correction Factor interactive graphs

The interactive graphs function by allowing users to select the data they wish to plot. Upon selecting their preferred dataset/s (Given Values, Duffie and Beckmann and Spencer's), the system generates a corresponding graph. This approach enables users to select from all three datasets, facilitating easy comparison of deviations between them.

4.3.7 Discussion

Values of the eccentricity correction factor for each day of the year taken from a python file named ecf_data. A graph for this data was plotted and a table was generated sourced from "An Introduction to Solar Radiation" [12]. Then the codes for calculating, plotting, and generating tables of the eccentricity correction factor using the two expressions namely; Spencer's and Duff and Beckmann were created. Thereafter, a graphical comparison of the three graphs was made and, the root mean square error and percentage error for the two expressions in relation to the values obtained from "An Introduction to Solar Radiation" [12]. For equation (4.3.5) the root mean square error was found to be 0.00168 and the average percentage error was 0.14%. The root mean square error obtained from equation (4.3.3) was 4.91×10^{-5} and there was no error obtained from this expression.

4.3.8 Conclusion

There was no significant deviation observed from the expression done by Spencer as depicted in equation (4.3.3) expression and the percentage error stood at 0.00%. This could have meant that the results shown in "An Introduction to Solar Radiation" [12] might have emanated from equation (4.3.3). Both expressions can be used, for greater accuracy and precision Spencer's expression is preferred while Duffie and Beckmann's expression can be handy for obtaining quick solutions. The highest values were observed between December and January while the lowest occurred in July.

Interactive graphs created show the deviation of the values of the two expressions with the values obtained from "An Introduction to Solar Radiation" [12] The interactive graph applet can be used in education and research to evaluate and select the most appropriate equations for specific applications.

4.4 The Solar Declination

4.4.1 Theory

The declination angle is a concept that represents the angle between the rays of the Sun and the plane of the Earth's equatorial plane. It varies throughout the year due to the Earth's axial tilt with respect to its orbit around the Sun. The Earth revolves around the Sun in an elliptical orbit and completes one revolution in 365 days or once in a year. It completes one rotation around its polar axis in approximately 24 hours relative to the Sun. This polar axis is inclined at an angle of approximately 23.45 degrees from the plane of its orbit around the Sun. Roughly, it ranges from -23.45 degrees to 23.45 degrees. At the vernal and autumn equinoxes the declination angle is equal to 0^{o}. Day and night cycles, as well as seasonal variations, are a result of the Earth's rotations and its tilt away from the plane of its orbit around the Sun.

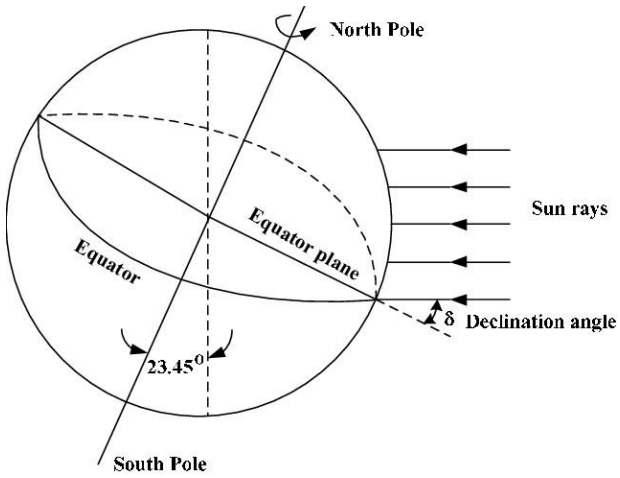


Figure 22: Declination Angle

[30]

Various formulations have been developed to compute the declination angles for each day of the year, including expressions such as Spencer's, Perrin de Brichambaut's, and Cooper's.

Spencer's expression is given by:

$$\delta = (0.006918 - 0399912\cos\Gamma + 0.070257\sin\Gamma - 0.006758\cos 2\Gamma + 0.000907\sin 2\Gamma - 0.002697\cos 3\Gamma + 0.00148\sin 3\Gamma)\left(\frac{180}{\pi}\right) \quad (4.4.1)$$

Perrin de Brichambaut is given by:

$$\delta = \cos^{-1}\left\{0.4\sin\left[\frac{360}{365}(d_n - 82)\right]\right\} \quad (4.4.2)$$

Cooper's expression is defined as:

$$\delta = 23.45\sin\left[\frac{360}{365}(d_n + 284)\right] \quad (4.4.3)$$

[12]

Where, δ represents the declination angle, Γ is the day angle, and d_n represents the day number, e.g. d_n = 1 for January 1, $d_n = 64$ for March 5. Day angle is explained in the previous chapter on page 54.

4.4.2 Using data from "An Introduction to Solar Radiation" [12]

Like in the previous chapter, data for declination angles from "An Introduction to Solar Radiation" [12] was used. A Python script named tabulated_declination_data with the list of declination angles saved as tabulated_declination_list was created and saved in the same folder as the Python script that was used to create the graph and the table for declination angles. After executing the computer program, it produced the output depicted in Figure 23 and table shown in appendix 4 on pages 121-123. Figure 23 illustrates the graph of declination angles for each day of the year [12].

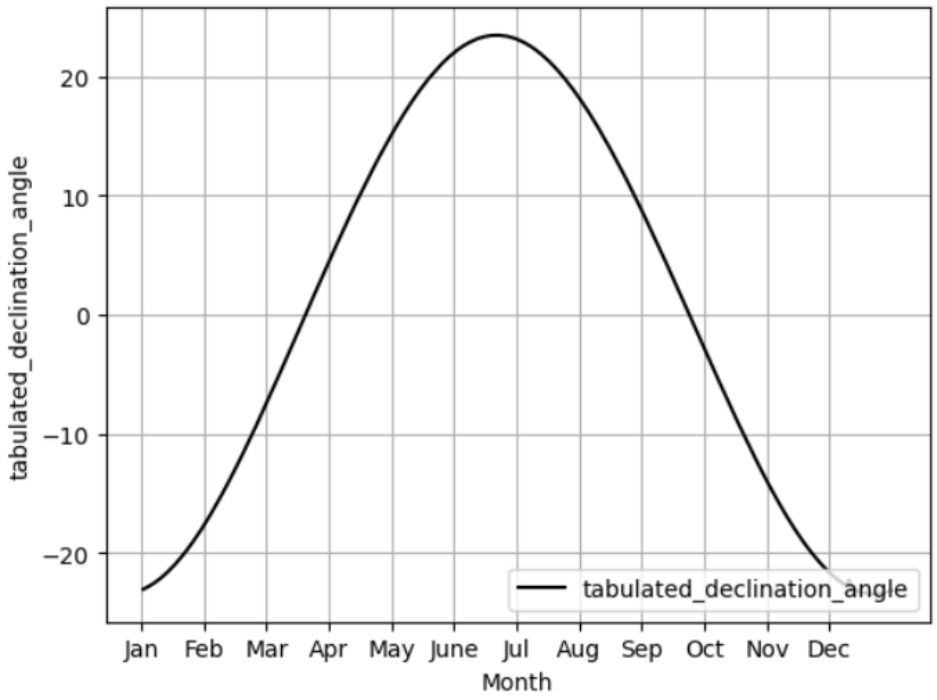


Figure 23: Declination Angles

4.4.3 Calculating Declination Angles using Spencer's expression

Employing Spencer's expression as shown in equation (4.1), a computer code (appendix 4 pages 124-126) was developed to compute and generate graphical and tabular data of declination angles for each day of the year. The code produced the results displayed in Figure 24 and the table on page 127-129.

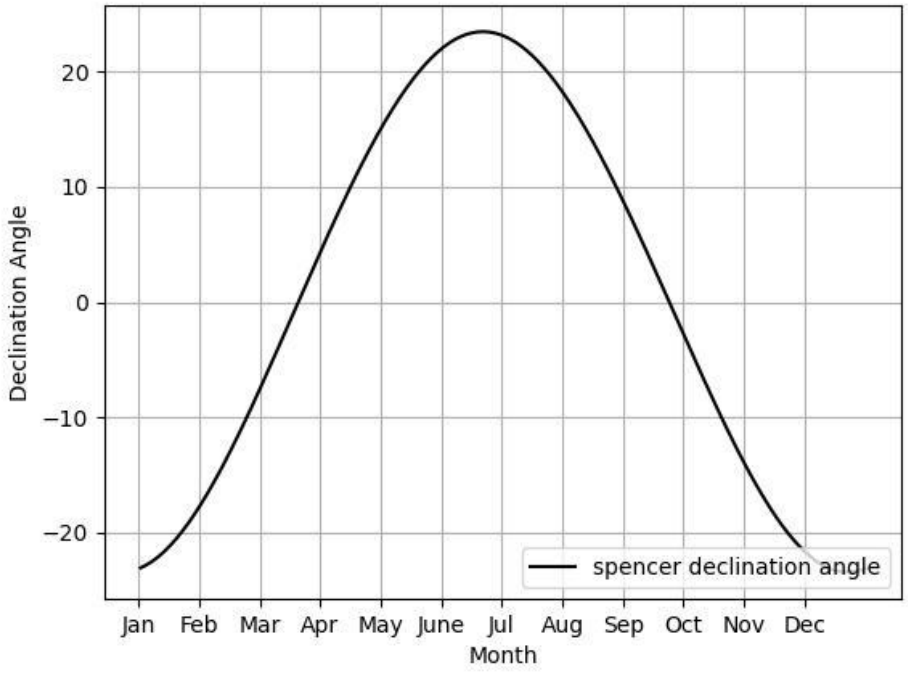


Figure 24: Declination Angles from Spencer's expression

4.4.3.1 Root Mean Square Error for Spencer's expression

Using equation (4.1.7), a computer code was developed to compute the root mean square error between declination angles derived from Spencer's expression and those obtained from "An Introduction to Solar Radiation" [12]. Upon execution, the program calculated a root mean square error of 0.0777. Additionally, another computer program was created to calculate the average percentage error using equation (4.1.9), resulting in an average percentage error of 0.66%.

4.4.4 Calculating Declination Angles using Perrin de Brichambaut's expression

Using Perrin de Brichambaut's formula described in equation (4.4.2), a computer program (appendix 4 pages 134-136) was developed to calculate and generate graphical and tabular representations of declination angles for each day of the year. The program produced the results displayed in Figure 25 and the corresponding table on page 137-139, the variation of the declination angles for each day of the year.

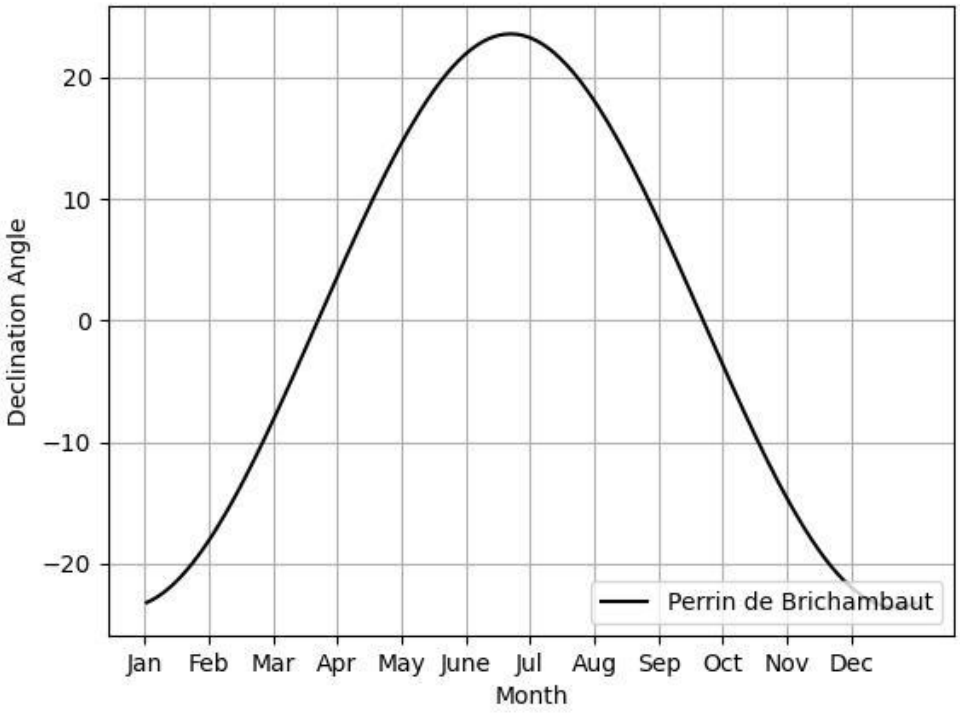


Figure 25: Declination Angles using Perrin de Brichambaut

4.4.4.1 Root Mean Square Error for Perrin de Brichambaut's expression

The root mean square error for the declination angles, comparing those calculated using Perrin de Brichambaut's expression to those obtained from "An Introduction to Solar Radiation" [12] was computed and found to be 0.5049. Moreover, the average percentage error was 4.40%

4.4.5 Calculating Declination Angles using Cooper's expression

Using Cooper's formula described in equation (4.4.3), a computer program (appendix 4 pages 141-143) was created to calculate and generate graphical and tabular representations of declination angles for each day of the year. The program generated results shown in Figure 26 and a corresponding table on page 144-146, illustrating the declination angles.

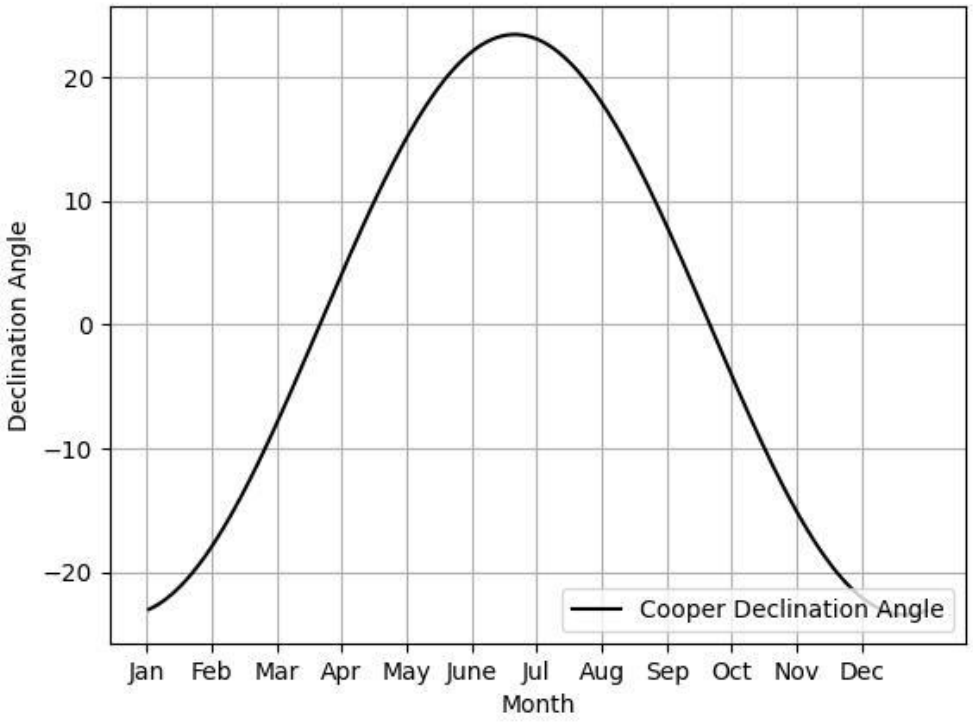


Figure 26: Declination Angles using Cooper's expression

4.4.5.1 Root Mean Square Error for Cooper's expression

The root mean square error of the declination angles between those calculated using Cooper's expression and those obtained from "An Introduction to Solar Radiation" [12] was found to be 0.586. Additionally, the average percentage error was determined to be 1.63%

4.4.6 Comparison of the Graphs

Subsequently, a graphical visual comparison was conducted between the values derived from equations (4.4.1), (4.4.2), (4.4.3) and those from "An Introduction to Solar Radiation" [12] as illustrated in Figure 27.

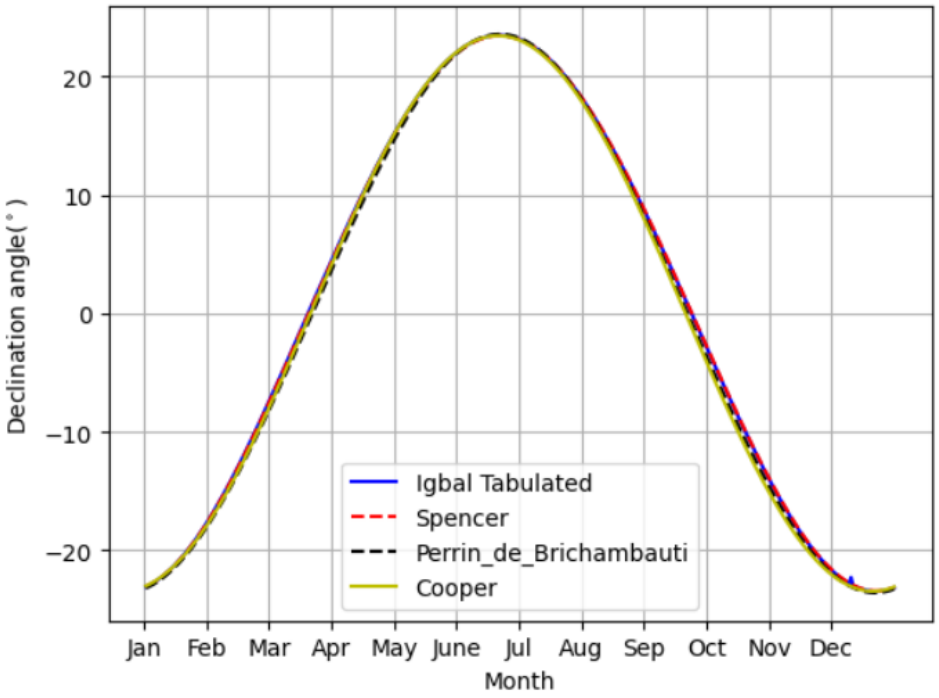


Figure 27: Comparison of the four graphs

4.4.7 Declination Interactive Graphs

To enhance interactivity and incorporate a graphical user interface, a computer code was developed using Ipywidgets. Figure 28 demonstrates the functionality of the applet designed for calculating declination angles.

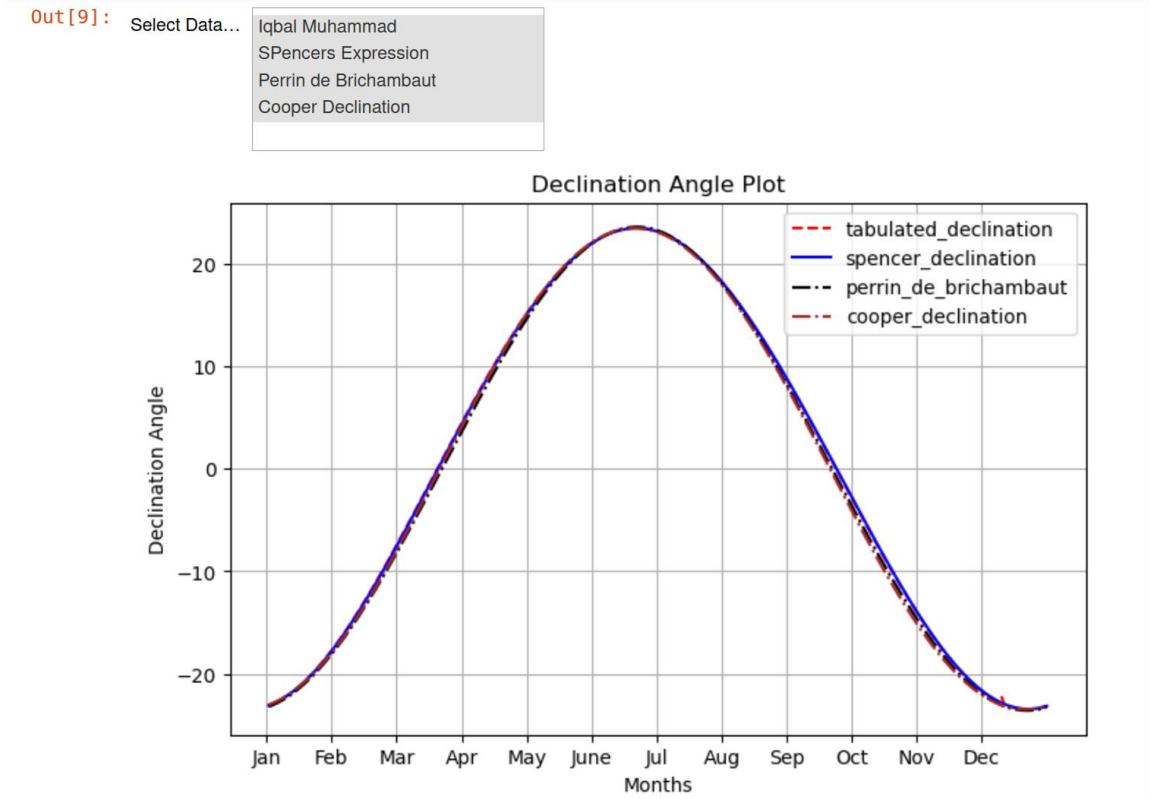


Figure 28: Interactive graphs for Declination Angles

The interactive graphs let users choose the data they want to see plotted. After selecting their desired data, it's displayed as a graph. This setup allows users to pick from all four datasets (Spencer's, Cooper's, Perrin de Brichambaut's and data from “An Introduction to Solar Radiation” [12], making it simple to compare differences among them.

4.4.8 Discussion

In investigating the impact of minimum and maximum declination angles on solar energy, three distinct expressions namely; Spencer's, Cooper's, and Perrin de Brichambaut's were employed to calculate the declination angle for each day of the year. The evaluation of these expressions revealed varying degrees of accuracy and precision. Spencer's expression exhibited the most favorable performance, yielding a root mean squared error of 0.0777 and an average percentage error of 0.66%. Cooper's expression followed with a root mean square error of 0.586 and a percentage error of 1.63%, while Perrin de Brichambaut's expression demonstrated a root mean square error of 0.505 and a percentage error of 4.40%.

4.4.9 Conclusion

Using Spencer's expression, the lowest declination angle was found to be -23.45° on December 22, while the highest declination angle was 23.46° on July 22. These findings underscore the significance of accurate declination angle calculations for each day of the year. Large positive declination angles are observed between April and August. During this period locations in the northern hemisphere experience longer daylength hours. This results in increased solar radiation while the opposite happens for locations in the southern hemisphere. Knowing the declination angle helps to check how changes in the tilt of the Earth's axis might affect how much solar energy can be harnessed at a particular time. Spencer's expression, which shows minimal errors, makes it a good choice when conducting research in solar energy studies. While other expressions can also be used when performing calculations.

It is also important to note from the graphs and tables that during equinoxes, on March 20/21 and September 22/23 the declination angle is almost zero resulting in equal daylengths and nights.

The applet offers a clear visualization of how different equations impact solar declination over time, allowing users to assess the performance of various models. This helps in determining which equation best aligns with empirical data or specific needs. By comparing these models, users can swiftly identify variations, enhancing their understanding of each model's accuracy and limitations.

4.5 Daylength

4.5.1 Theory

Daytime or Daylength is the period of the day during which a specific place receives light from direct sunlight. This happens when the sun is visible above the horizon or when a place is facing the sun. It can also be defined as the duration of the period between sunrise and sunset [31]. Daylength is dependent on the geographic latitude of a place and the inclination of the sun. Hence, it varies throughout the year and from place to place.

In this section, we will investigate the effects of latitudes on the daylengths on different locations.

Daylength is determined by the following expression:

$$D_L = \frac{2}{15}\cos^{-1}(\tan\varphi\tan\delta) \tag{4.5.1}$$

Where φ is the geographical latitude and δ is the declination angle [12]. Latitude is the angular distance between the equator and a particular location.

4.5.2 Calculating the Daylength

The computer code (appendix 5 pages 146-148) which outlines the process of using the latitude as the input to calculate the daylength values for any location was generated using equation (4.5.1). This was done by, first, creating a function. Then, the function was called upon to calculate the daylength for any particular latitude ranging from $-90°$ to $90°$. For places in the northern hemisphere latitudes are taken to be positive while for locations in the southern hemisphere they are taken to be negative.

4.5.2.1 Daylengths for Lusaka

The daylength for each day of the year for Lusaka which is at latitude 15° south was calculated and an output was given out inform of a graph and a table as shown in Figure 29 and a table in appendix 5 on pages 149-151.

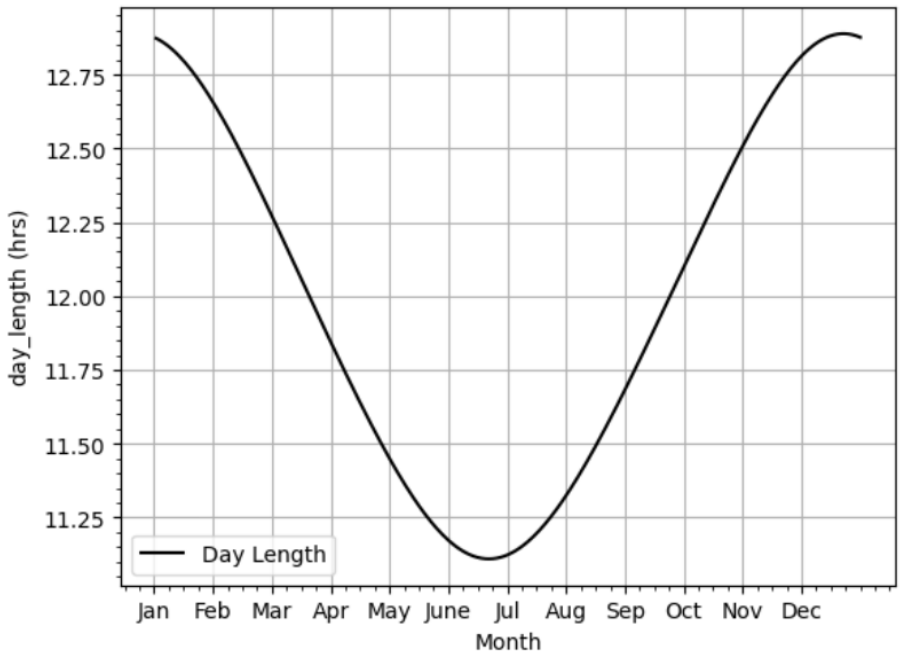


Figure 29: Graph of Daylength for Lusaka

4.5.2.2 Daylengths for Greenland

Next, the daylength for Greenland which is located at latitude 71.7069°N was calculated and the results are presented in both a graph (Figure 30) and a table (appendix 5 on pages 152-154).

Figure 30 shows how the daylength for Greenland varies throughout the year.

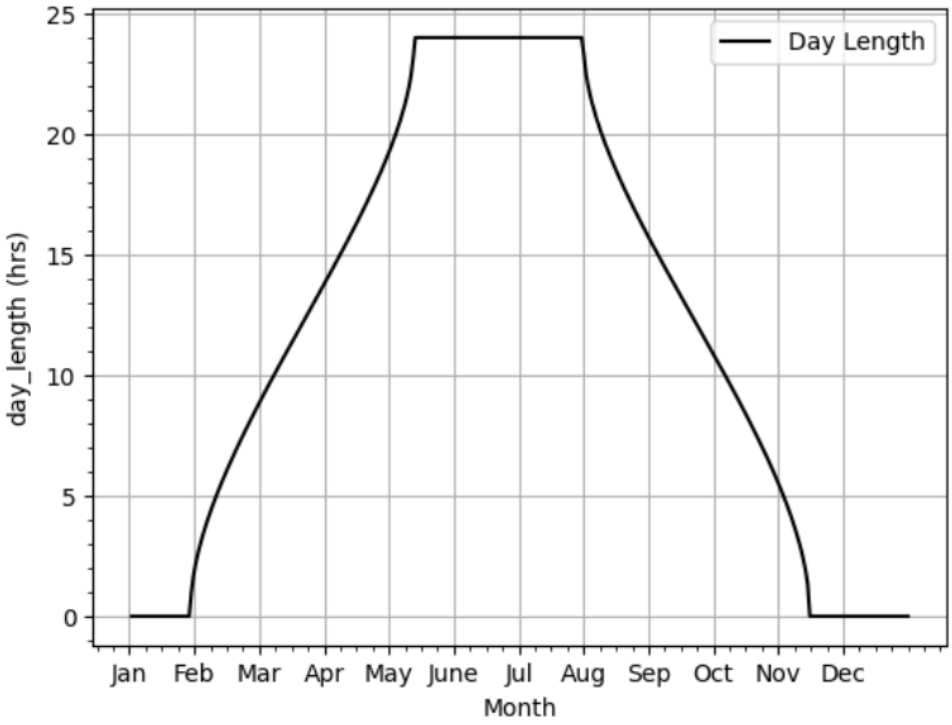


Figure 30: Graph of Daylength for Greenland

4.5.2.3 Daylengths for Dome Charlie

Dome Charlie which is at latitude 75.1000°S its daylengths are shown in Figure 31 and tabulated in appendix on pages 155-157.

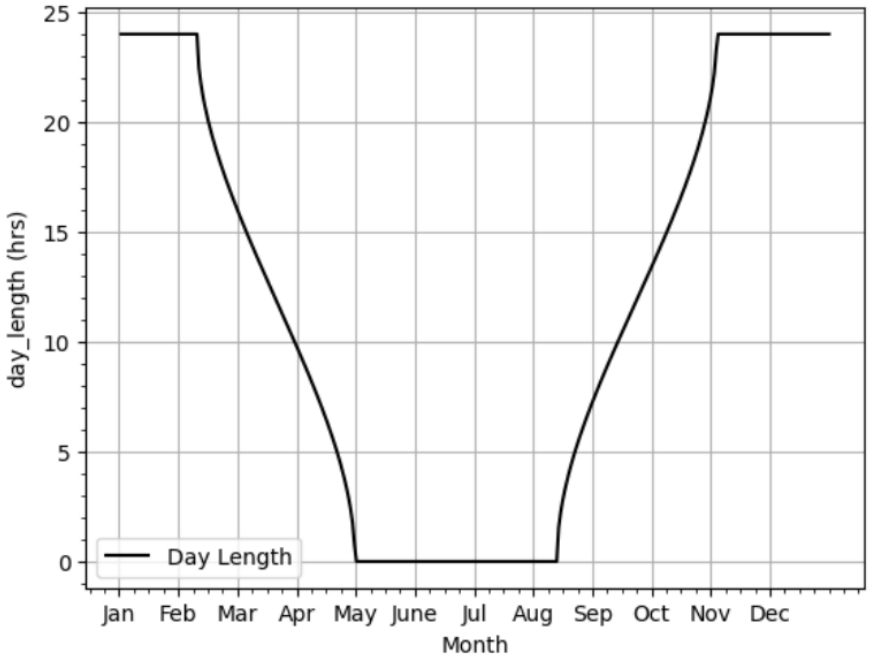


Figure 31: Graph of Daylength for Dome Charlie

4.5.2.4 Daylengths for Nanyuki

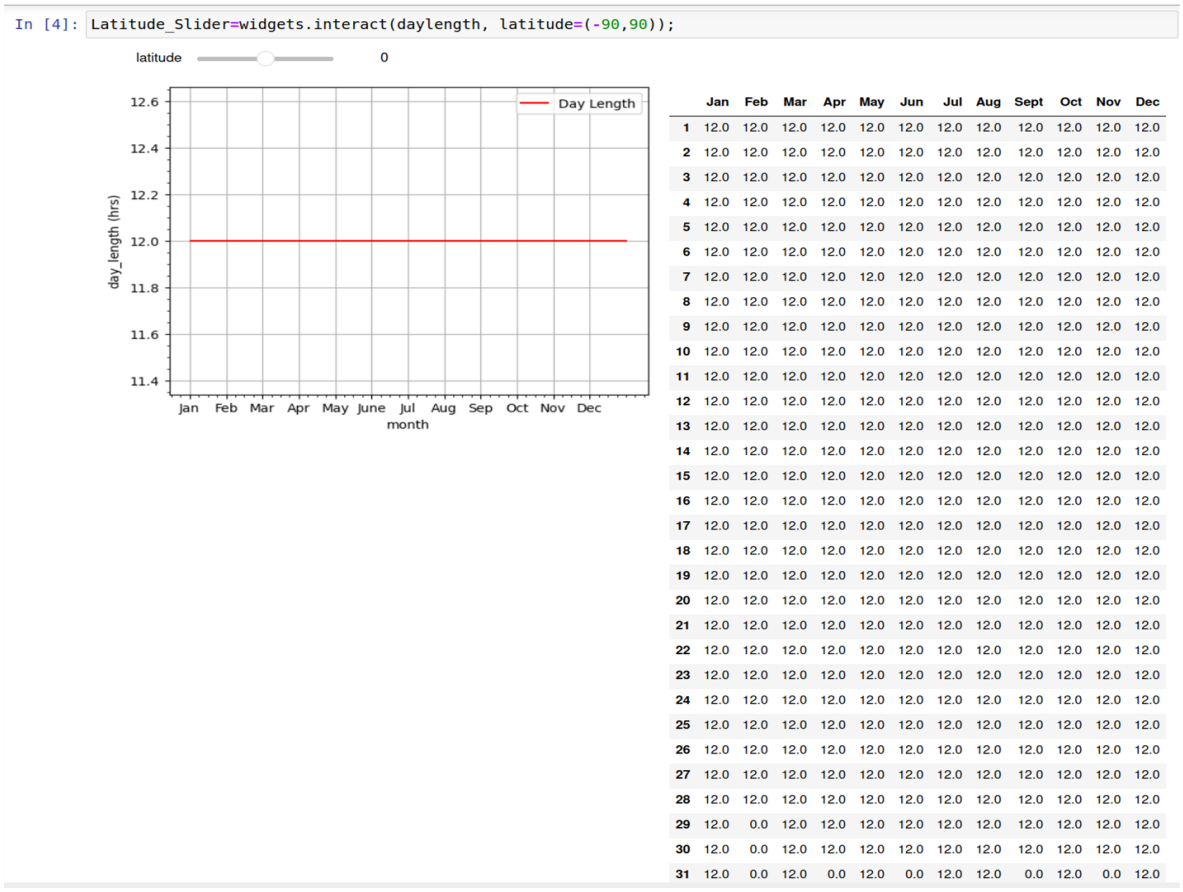
Another location that was considered was Nanyuki a place in Kenya which is at latitude 0.0074°N. Using the same procedure that was used in the previous section, the daylength was calculated and presented in tabular format in appendix 5 on pages 158-160. As can be observed this place is situated along the equator, hence, it maintains a consistent daylength of 12:00 hours for all the days.

4.5.3 Daylength Interactive Graphs

To include interactivity and create a graphical user interface, a computer code for calculating daylengths of any location was developed in Jupyter Notebook using Ipywidgets. The widgets created outputs both graphical and tabular data in real time.

4.5.3.1 Slider widget

A slider widget that allows users to input latitude values from -90° to 90° was created as shown in Figure 32.



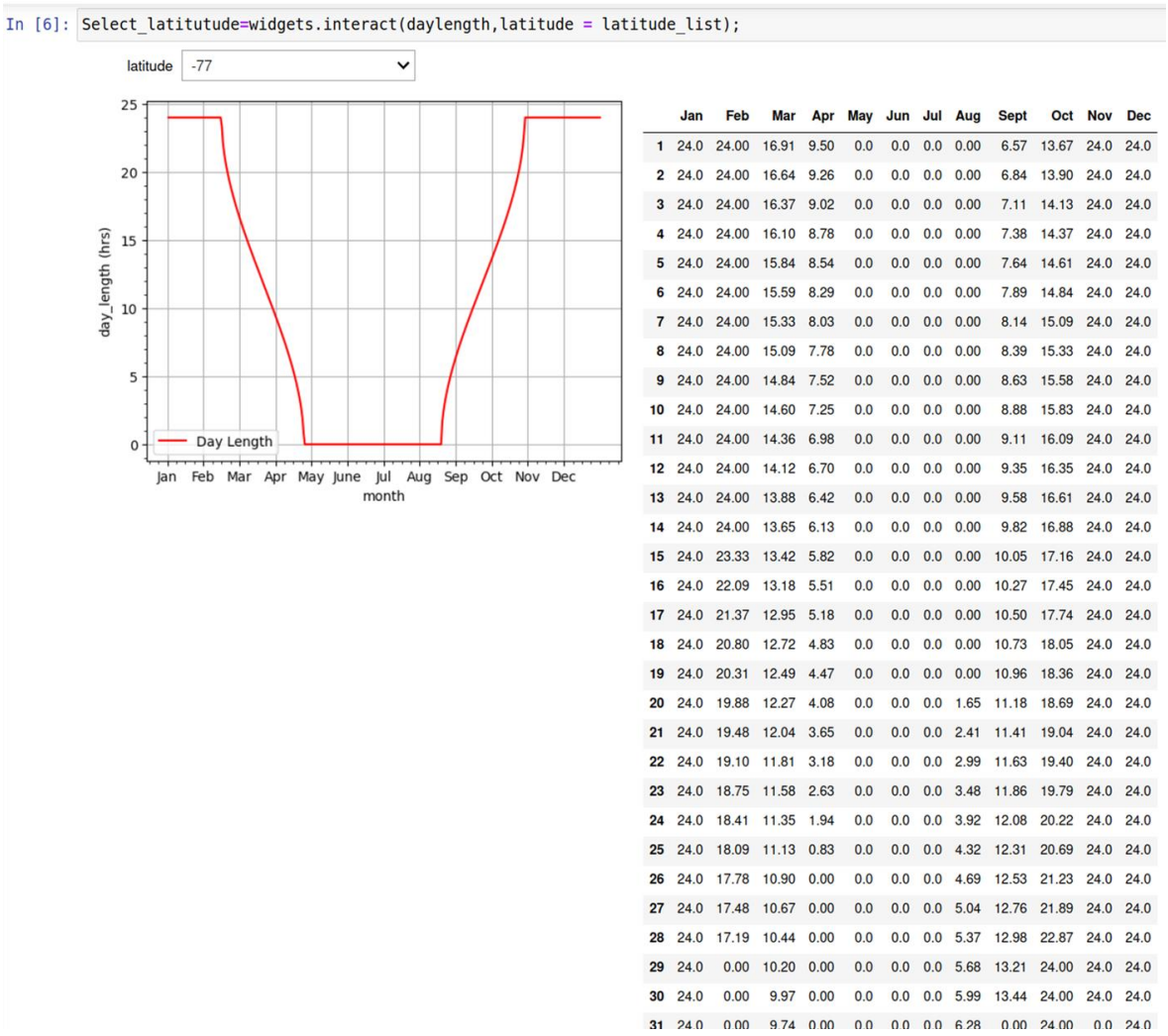
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sept	Oct	Nov	Dec
1	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
2	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
3	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
4	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
5	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
6	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
7	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
8	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
9	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
10	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
11	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
12	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
13	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
14	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
15	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
16	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
17	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
18	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
19	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
20	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
21	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
22	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
23	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
24	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
25	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
26	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
27	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
28	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
29	12.0	0.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
30	12.0	0.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
31	12.0	0.0	12.0	0.0	12.0	0.0	12.0	12.0	0.0	12.0	0.0	12.0

Figure 32: Daylength slider widget in use

The slider is employed by the user to alter latitudes, with each adjustment leading to updates in both the graph and the table.

4.5.3.2 Daylength dropdown latitude select widget

Next, a widget with a dropdown menu of latitudes from -90° to 90° was created. Figure 33 shows the widget with a dropdown menu where the user can choose their desired menu with latitudes ranging from -90° to 90° in steps of 1°.



	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sept	Oct	Nov	Dec
1	24.0	24.00	16.91	9.50	0.0	0.0	0.0	0.00	6.57	13.67	24.0	24.0
2	24.0	24.00	16.64	9.26	0.0	0.0	0.0	0.00	6.84	13.90	24.0	24.0
3	24.0	24.00	16.37	9.02	0.0	0.0	0.0	0.00	7.11	14.13	24.0	24.0
4	24.0	24.00	16.10	8.78	0.0	0.0	0.0	0.00	7.38	14.37	24.0	24.0
5	24.0	24.00	15.84	8.54	0.0	0.0	0.0	0.00	7.64	14.61	24.0	24.0
6	24.0	24.00	15.59	8.29	0.0	0.0	0.0	0.00	7.89	14.84	24.0	24.0
7	24.0	24.00	15.33	8.03	0.0	0.0	0.0	0.00	8.14	15.09	24.0	24.0
8	24.0	24.00	15.09	7.78	0.0	0.0	0.0	0.00	8.39	15.33	24.0	24.0
9	24.0	24.00	14.84	7.52	0.0	0.0	0.0	0.00	8.63	15.58	24.0	24.0
10	24.0	24.00	14.60	7.25	0.0	0.0	0.0	0.00	8.88	15.83	24.0	24.0
11	24.0	24.00	14.36	6.98	0.0	0.0	0.0	0.00	9.11	16.09	24.0	24.0
12	24.0	24.00	14.12	6.70	0.0	0.0	0.0	0.00	9.35	16.35	24.0	24.0
13	24.0	24.00	13.88	6.42	0.0	0.0	0.0	0.00	9.58	16.61	24.0	24.0
14	24.0	24.00	13.65	6.13	0.0	0.0	0.0	0.00	9.82	16.88	24.0	24.0
15	24.0	23.33	13.42	5.82	0.0	0.0	0.0	0.00	10.05	17.16	24.0	24.0
16	24.0	22.09	13.18	5.51	0.0	0.0	0.0	0.00	10.27	17.45	24.0	24.0
17	24.0	21.37	12.95	5.18	0.0	0.0	0.0	0.00	10.50	17.74	24.0	24.0
18	24.0	20.80	12.72	4.83	0.0	0.0	0.0	0.00	10.73	18.05	24.0	24.0
19	24.0	20.31	12.49	4.47	0.0	0.0	0.0	0.00	10.96	18.36	24.0	24.0
20	24.0	19.88	12.27	4.08	0.0	0.0	0.0	1.65	11.18	18.69	24.0	24.0
21	24.0	19.48	12.04	3.65	0.0	0.0	0.0	2.41	11.41	19.04	24.0	24.0
22	24.0	19.10	11.81	3.18	0.0	0.0	0.0	2.99	11.63	19.40	24.0	24.0
23	24.0	18.75	11.58	2.63	0.0	0.0	0.0	3.48	11.86	19.79	24.0	24.0
24	24.0	18.41	11.35	1.94	0.0	0.0	0.0	3.92	12.08	20.22	24.0	24.0
25	24.0	18.09	11.13	0.83	0.0	0.0	0.0	4.32	12.31	20.69	24.0	24.0
26	24.0	17.78	10.90	0.00	0.0	0.0	0.0	4.69	12.53	21.23	24.0	24.0
27	24.0	17.48	10.67	0.00	0.0	0.0	0.0	5.04	12.76	21.89	24.0	24.0
28	24.0	17.19	10.44	0.00	0.0	0.0	0.0	5.37	12.98	22.87	24.0	24.0
29	24.0	0.00	10.20	0.00	0.0	0.0	0.0	5.68	13.21	24.00	24.0	24.0
30	24.0	0.00	9.97	0.00	0.0	0.0	0.0	5.99	13.44	24.00	24.0	24.0
31	24.0	0.00	9.74	0.00	0.0	0.0	0.0	6.28	0.00	24.00	0.0	24.0

Figure 33: Daylength dropdown latitude select widget in use

4.5.3.3 Multi-Select widget

Finally, an interactive applet was created, this enabled users to observe daylength variations across different locations. This feature allows users to explore how latitude influences daylength throughout the year for each specific location The specified locations include Lusaka with a latitude of 15°S, Cairo at 30.0444°S, Greenland situated at 71.7069°N, Dome Charlie positioned at 75.1000°S, and Nanyuki located along the equator at 0.0074°N. Using the multi-select widget users can choose places and make plots to compare daylengths.

Figure 34 shows how the daylengths of different locations vary for each day of the year.

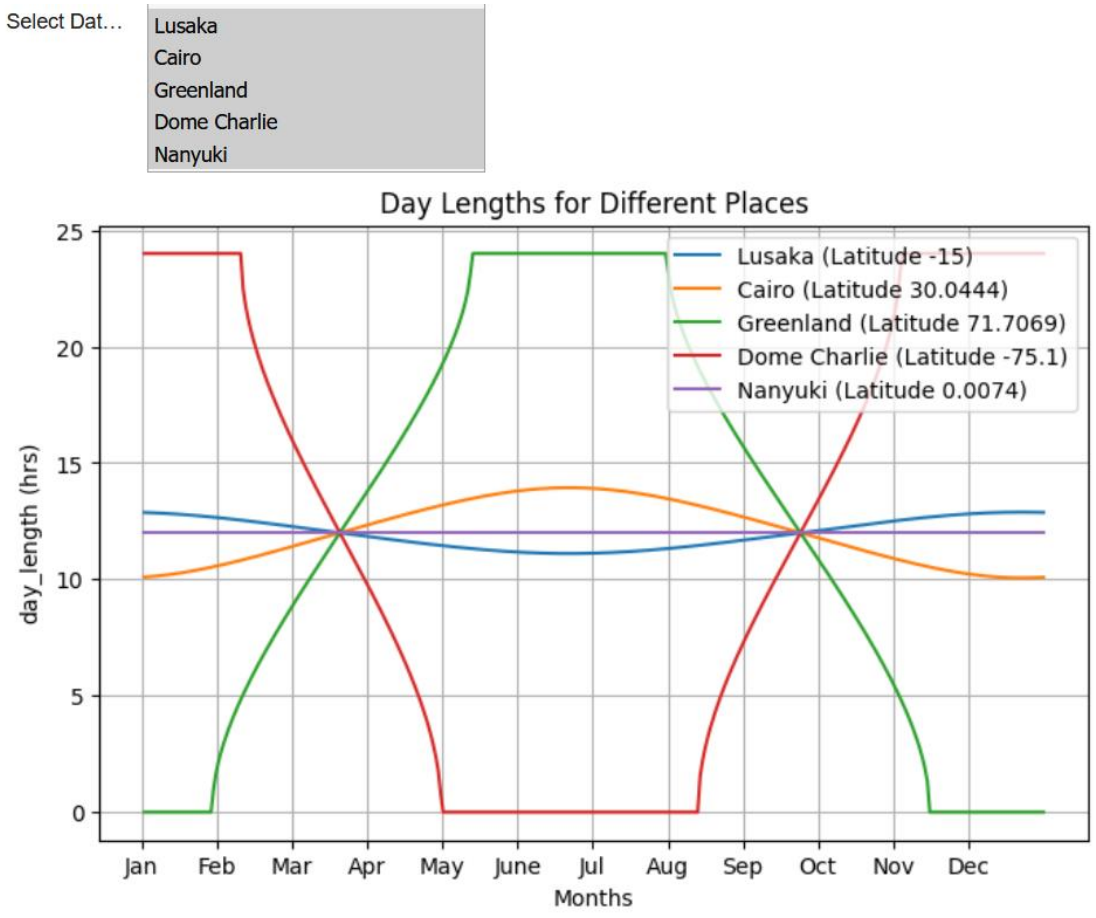


Figure 34: Daylength for different cities plotted using multi select widget in use

4.5.4 Discussion

After running the computational algorithms, notable variations were observed in daylengths for Lusaka, spanning from 11.11 hours to 12.89 hours. Greenland has a unique solar pattern, with no

sunlight from November 15 to January 29, where each day remains consistently 0.00 hours long, hence, there is no sunrise. On the contrary, from May 14 to July 31, it receives continuous sunlight, with daylengths constantly at 24 hours, so during this period the sun does not set. In contrast, Dome Charlie experiences continuous sunlight from November 4 to February 8, where each day's daylength extends to 24 hours, therefore, there is no sunset at this time. But, from May 2 to August 12, Dome Charlie encounters continuous darkness as the daylength for each day remains at 0 hours, causing the sun to remain below the horizon. Meanwhile, Nanyuki, situated along the equator, maintains a consistent daylength of 12:00 hours throughout the year.

The applet was compiled into a standalone executable file for easy installation and use, eliminating the need for coding expertise or source code access. Figure 35 shows the installed Daylength Multi-Place Plotter applet in use. The applet works in such a way that the user enters the place of their choice and its latitude. It takes more than one input as it makes graphical comparison of two or more places. In Figure 33 two places were chosen, Cairo which is on the northern part of the equator with the latitude of 30.03333° and Cape Town which is on the southern part with the latitude of 33.918861°. The two graphs meet at the two equinoxes (March 20/21 and September 22/23) where the daylength is exactly 12:00 hours. Between this period Cairo experiences daylengths of more than 12:00 hours while the opposite happens in Cape Town. So, when it is summer on the northern part of the equator, the southern part experiences winter.

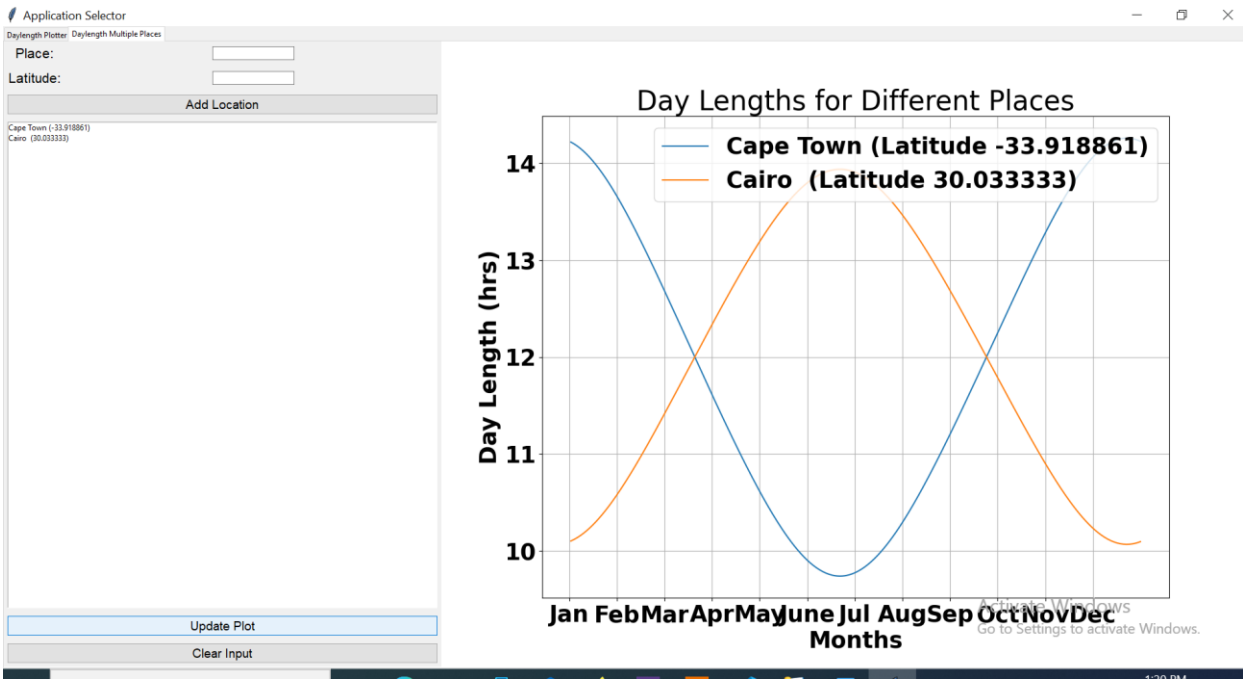


Figure 35: Daylength Multi-Places Plotter applet

4.5.5 Conclusion

We can see that daylengths change throughout the year because of the sun's position, which is called the declination angle. It also varies depending on where you are, mainly because of differences in latitudes. Near the equator, like in Lusaka, days are close to 12 hours long. At the equator itself, like in Nanyuki, Kenya, where the latitude is 0.0074°N, the length of the day stays exactly 12 hours all year round. In the northern hemisphere, during winter when the sun is low in the sky ($cos\ \omega_s \geq 1$), some places don't get any direct sunlight. But during summer when the sun is high ($\omega_s \leq -1$), places like Greenland get sunlight all day long. In the southern hemisphere during the same time, places like Dome Charlie experience the opposite, with longer days in summer and shorter days in winter.

The applet created has a wide range of applications across various fields. In astronomy, it can determine the best time for observing celestial objects. For agriculture, farmers can use daylength data to plan when to plant, predict flowering times, and care for plants that need specific light conditions. Travelers can benefit from the applet by understanding daylight hours at different locations and plan their activities accordingly. In education and research, the applet can be used to enhance students' understanding of how factors such as the rotation and orbit of the Earth,

declination angle and latitude affect daylengths and seasons. Additionally, it is valuable for climate change research, as daylength data can be used to study environmental changes and their effects on ecosystems. Finally, in energy management, knowing daylength can help optimize solar irradiance use, leading to more efficient solar systems and energy consumption in buildings.

4.6 Extraterrestrial Solar Radiation on a Horizontal Plane

4.6.1 Background

Extraterrestrial radiation refers to the radiation that the Earth receives from the Sun [32]. It comes from outside the Earth's atmosphere and is the primary source of energy for the Earth and plays a crucial role in determining the climatic conditions. The equation for the daily radiation on a horizontal surface is given by:

$$H_0 = \frac{24}{\pi}\ x\ 1367\ x\ E_0\left[\left(\frac{\pi}{180}\right)\omega(sin\delta sin\varphi) + (cos\delta cos\varphi sin\omega)\right] \qquad (4.6.1)$$

Where: H_0 is the daily extraterrestrial radiation, E_0 is the eccentricity correction factor, ω is the hour angle, δ is the declination angle, $\varphi$is the latitude [12].

4.6.2 Calculating the Daily Extraterrestrial Radiation for Lusaka

Using equation (4.6.1) a computer code as shown in appendix 6 on page 161-163 for calculating daily extraterrestrial radiation on a horizontal surface for any location was generated.

Employing the computer code, the daily extraterrestrial radiation solar radiation on a horizontal surface for each day of the year for Lusaka with latitude -15° was calculated. The output gave a plot as shown in Figure 36 and a table of values depicted in appendix 5 on page 164-166. The minimum daily extraterrestrial radiation solar radiation received was found to be $25.56\ \mathrm{MJ}/m^2$ while the maximum observed was $41.01\ \mathrm{MJ}/m^2$.

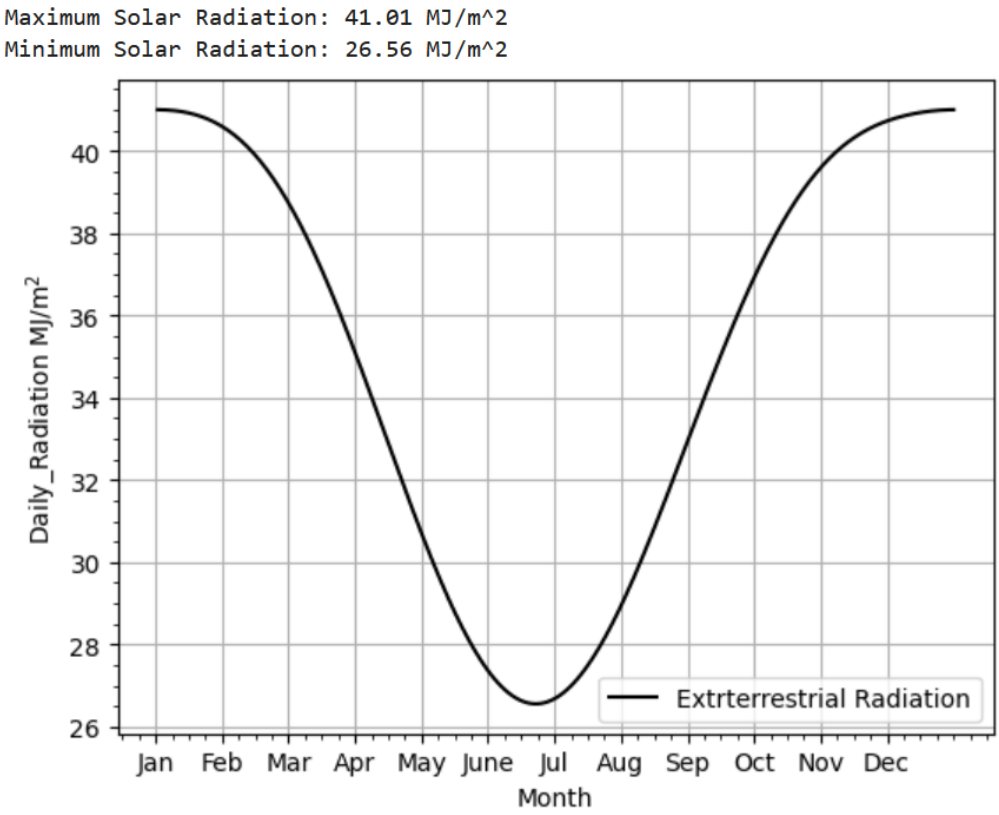


Figure 36: Graph of Daily Extraterrestrial Solar Radiation over Lusaka

4.6.3 Daily Extraterrestrial Solar Radiation Calculator

Next, the interactive applet for calculating the extraterrestrial solar applet was created as shown in Figure 37. It consists of the latitude and day number entry boxes where the user inputs the latitude of the location of their choice and day number. For the day number; January, 1 is taken as day number 1 and December 31st as day number 365. It also has two buttons, the calculate and reset buttons. Once the user enters the latitude and the number then they can press the calculate button to get the results. The reset button is used to clear the output and the entry boxes to pave way for new entries.

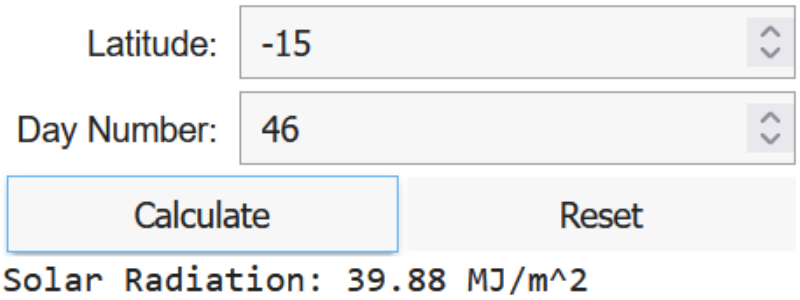


Figure 37: Daily Extraterrestrial Solar Radiation calculator in use

4.6.4 Hourly Extraterrestrial Solar Radiation Calculator

The hourly extraterrestrial radiation is calculated using equation (6.2) shown below.

$$I_0 = \frac{12}{\pi}\ x\ 1367\ x\ E_0\left[cos\varphi cos\delta(sin\omega_2 - sin\omega_1) + \left(\frac{\pi(\omega_2 - \omega_1)}{180}\right)(sin\delta sin\varphi)\right] \qquad (4.6.2)$$

where, ω_1 is the initial time and ω_2 is the final time.

Using the above equation, the applet for the hourly extraterrestrial radiation was created. Figure 38 shows the hourly extraterrestrial radiation calculator which works in a similar way as the daily extraterrestrial radiation calculator shown in Figure 37. It consists of the latitude, day number, initial and final time entry boxes, and two buttons; the calculate and reset buttons.

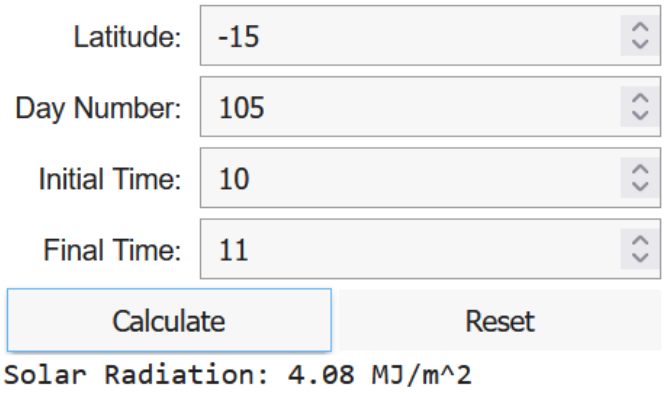


Figure 38: Hourly Extraterrestrial Radiation calculator in use

4.6.5 Monthly Average Daily Extraterrestrial Solar Radiation at different Latitudes

Knowing the monthly average extraterrestrial solar radiation at a particular location is essential for energy planning, research and development. Monthly average extraterrestrial solar radiations were calculated using equation (4.6.1). The following day numbers were used to calculate the average radiation for each month, from January to December, respectively: 17, 45, 74, 105, 135, 161, 199, 230, 261, 292, 322, and 347 [12]. The computer code for calculating and plotting the monthly average extraterrestrial radiation for latitudes from -90° to 90° in steps of 5° was developed as shown on pages 167-168. Using the developed computer program, graphs for monthly average daily extraterrestrial radiation were plotted as shown in Figure 39 and tabulated values are shown in appendix 6 on page 169-171.

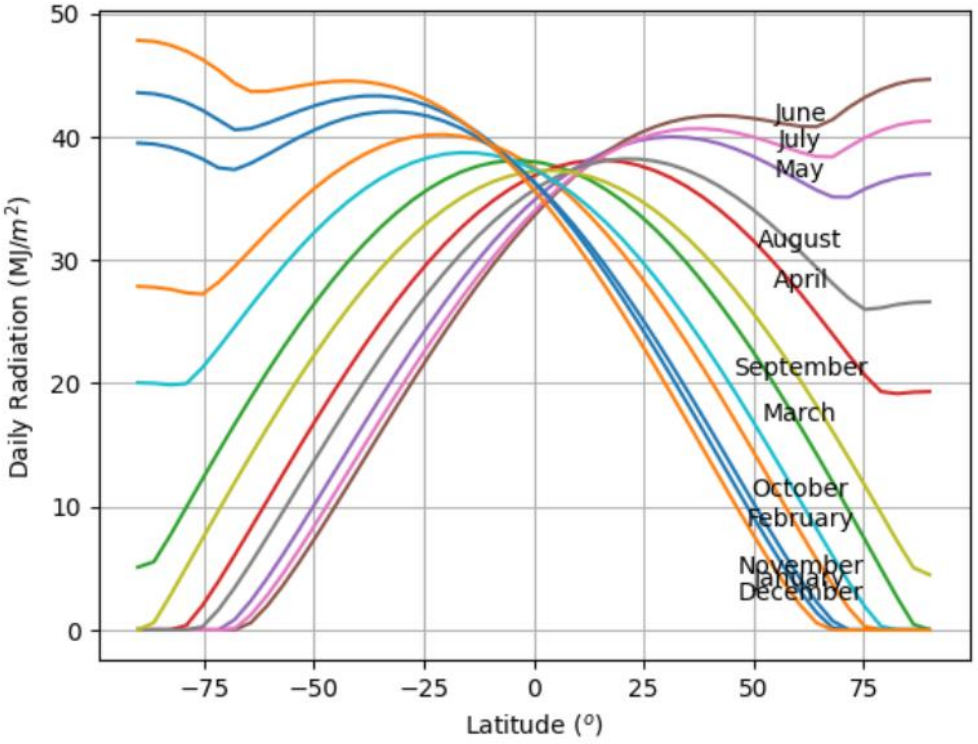


Figure 39: Monthly Average Daily Extraterrestrial Solar Radiation for all the months

4.6.6 Discussion

Based on equation (4.6.1), a Python program for calculating the daily extraterrestrial radiation for each day of the year reaching any particular location on Earth was developed. This program has a function with latitude as the key word argument. After running the program, the user calls on the function and enters the latitude of their choice. The output given out is the graph and the table showing the variation of the daily extraterrestrial radiation of the whole year for that particular location.

As an example, for Lusaka with a latitude of -15°, higher extraterrestrial daily radiations were recorded in the months of December/January, and lower extraterrestrial radiation was recorded in the months of June/July. The highest was on 31st December, which stood at $41.01\ \mathrm{MJ/m^2}$day, while the lowest was $26.56\ \mathrm{MJ/m^2}$, which happened on 23rd July.

To create an interactive program, Ipywidgets were integrated into the program. Two input widgets were created where the user enters the latitude for any location and any day number of the year to calculate the extraterrestrial radiation for that particular day. After entering these variables, the user then presses the calculate button to get the results. As demonstrated in Figure 37 at latitude 15°S and a day number of 46, which is 15th February, the solar radiation received on this particular day is $39.88\ \mathrm{MJ/m^2}$.

For the hourly extraterrestrial radiation, four input widgets, namely Latitude, Day Number, Initial Time, and Final Time, were created. Once the user enters their variables, they then press the calculate button to get the hourly extraterrestrial radiation between two lengths of time. As shown in Figure 38, the average extraterrestrial solar radiation recorded between 10:00 hours and 11:00 hours on the 15th April in Lusaka is $4.08\ \mathrm{MJ/m^2}$.

A code for calculating monthly averages of daily solar radiation at different latitudes from south to north was formulated. The latitudes ranged from -90° to 90° in steps of 5°. On the extreme north, where the latitudes are above 65°, the lowest values of average monthly daily radiation of up to $0\ \mathrm{MJ/m^2}$ starting from October to March. During this period, this part of the world experiences fewer or no sunlight hours. Starting from May to July, they experience the highest solar radiation with sunlight hours of up to 24:00 hours. On the extreme south, the opposite happens. From April to August, low solar radiations of as low as $0\ \mathrm{MJ/m^2}$ are experienced as sunlight hours go to as low as 0:00 hours. From November to February, a large amount of solar radiation is received due

to the fact that sunlight hours of up to 24:00 hours are experienced. In order to have a visual analysis of the effects of latitude on the amount of energy received at different time of year the graph of monthly average against latitude was plotted as shown in Figure 39.

The interactive programs were converted into Tkinter graphical user interface and then turned into an executable with the menu system on top. Figure 40 shows an installed applet in use.

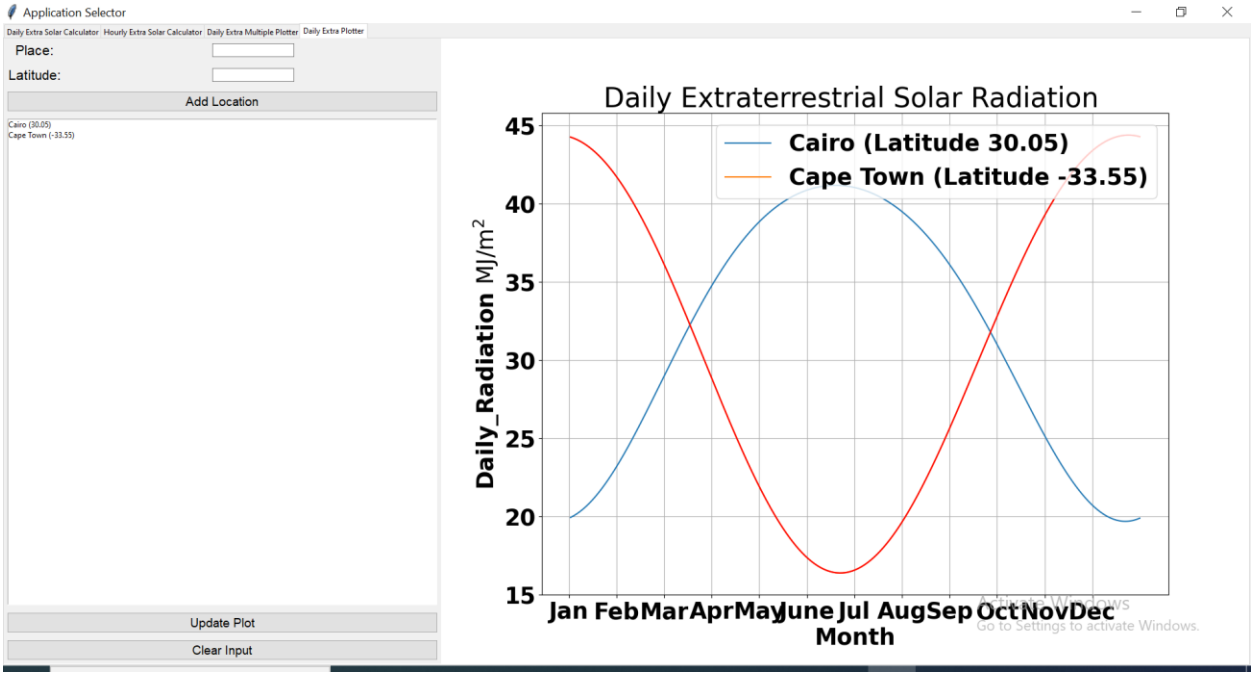


Figure 40: Extraterrestrial Radiation Applet

4.6.7 Conclusion

The applets created can be used to estimate, model and analyze solar radiation reaching a specific location on the earth's surface. This data is useful in climate studies, solar energy system design, agricultural planning, and environmental monitoring.

4.7 Reflection and Refraction at Dielectric Interfaces

4.7.1 Theoretical Background

The importance of dielectric interfaces in harnessing the most of radiation in solar energy technologies cannot be over-emphasized. The depths of the theoretical and practical understanding of reflection and refraction at these surfaces can lead to improved technological solar energy designs [33].

Reflection of light has a bearing on how much incident radiation will reach the photovoltaic cells. The greater the reflection coefficient the less the incident radiation and vice-versa. Refraction being the bending of light as it passes from one medium to another is important in the design of solar energy technologies for effective concentration of radiation on solar cells [34].

Reflection and refraction at dielectric interfaces are essential for boosting the effectiveness of photovoltaic (PV) cells. Using anti-reflection coatings reduces reflection between cell layers, maximizing light absorption. Textured surfaces and waveguiding structures trap light within the cell, increasing absorption efficiency. These techniques also enable selective absorption of specific wavelengths. In multi-junction cells, precise interface engineering optimizes light management across layers with different properties, enhancing overall efficiency. Leveraging these phenomena improves PV cells' ability to convert sunlight into electricity.

Reflection and refraction at dielectric interfaces can be understood using the Fresnel's equations. Light which is electromagnetic radiation is a transverse wave. Transverse waves are described in terms of polarization due to the orientation of the electric field vector. The following equations give the reflectance for the two polarization states; parallel and perpendicular:

$$r_{\parallel} = \left[\frac{{n_r}^2 cos\theta_i - n_i\sqrt{{n_r}^2 - {n_i}^2 sin^2\theta_i}}{{n_r}^2 cos\theta_i + n_i\sqrt{{n_r}^2 - {n_i}^2 sin^2\theta_i}}\right]^2 \quad (4.7.1)$$

and

$$r_{\perp} = \left[\frac{n_i cos\theta_i - \sqrt{{n_r}^2 - {n_i}^2 sin^2\theta_i}}{n_i cos\theta_i + \sqrt{{n_r}^2 - {n_i}^2 sin^2\theta_i}}\right]^2 \quad (4.7.2)$$

If we take it that incident solar energy is nearly unpolarized and equal amounts of energy are observed in each parallel and perpendicular polarization. The average reflection coefficient is given by:

$$\bar{r} = \frac{1}{2}(r_{\parallel} + r_{\perp}) \quad [25] \tag{4.7.3}$$

One of the most important ideas in optimizing solar energy techniques comes from the concept of Brewster's angle which is named after the physicist Sir David Brewster. It refers to the angle at which light with a specific polarization state passes through a transparent surface, such as glass or water, without any reflection. This occurs when the incident light is polarized perpendicular to the surface.

The determination of Brewster's angle can be derived from Snell's law, which relates the angles of incidence and refraction when light passes through different media. For light polarized parallel to the surface, there is no specific angle at which it is entirely transmitted without reflection. However, for light polarized perpendicular to the surface, Brewster's angle can be found by setting the reflected intensity to zero.

At Brewster's angle, the reflected light intensity becomes zero, and only the refracted light is observed. This phenomenon arises because the angle of refraction for the perpendicular polarized light is 90 degrees, causing it to travel along the surface without reflection.

The mathematical expression for Brewster's angle can be obtained by considering the relationship between the refractive indices of the incident and transmitting media [35].

4.7.2 Creating a Dielectric Interface input widget

The interactive plots were created by transforming the three equations given above into a computer program shown on pages 175-177. Figure 40 shows the applet in use. It works in such a way that the user enters the refractive indices of the two media indicated by index 1 and index 2 and clicks on the plot button to get the results. The reset button is used to clear current input and results to set the stage for new entries.

4.7.2.1 Air-Glass Interface

Let us assume that the user is interacting with an air-glass interface, where the refractive indices are 1 (air) and 1.5 (glass), respectively. After entering the indices, the user clicks on the plot button,

resulting in the generation of a graph. The applet displays Brewster's angle, denoting the point where reflectance is zero. In this specific case, it is observed at 56.27°, as illustrated in Figure 41. To modify the output, users can utilize the reset button and input their preferred refractive indices.

Figure 41 shows the output of the reflectance applet at dielectric interface

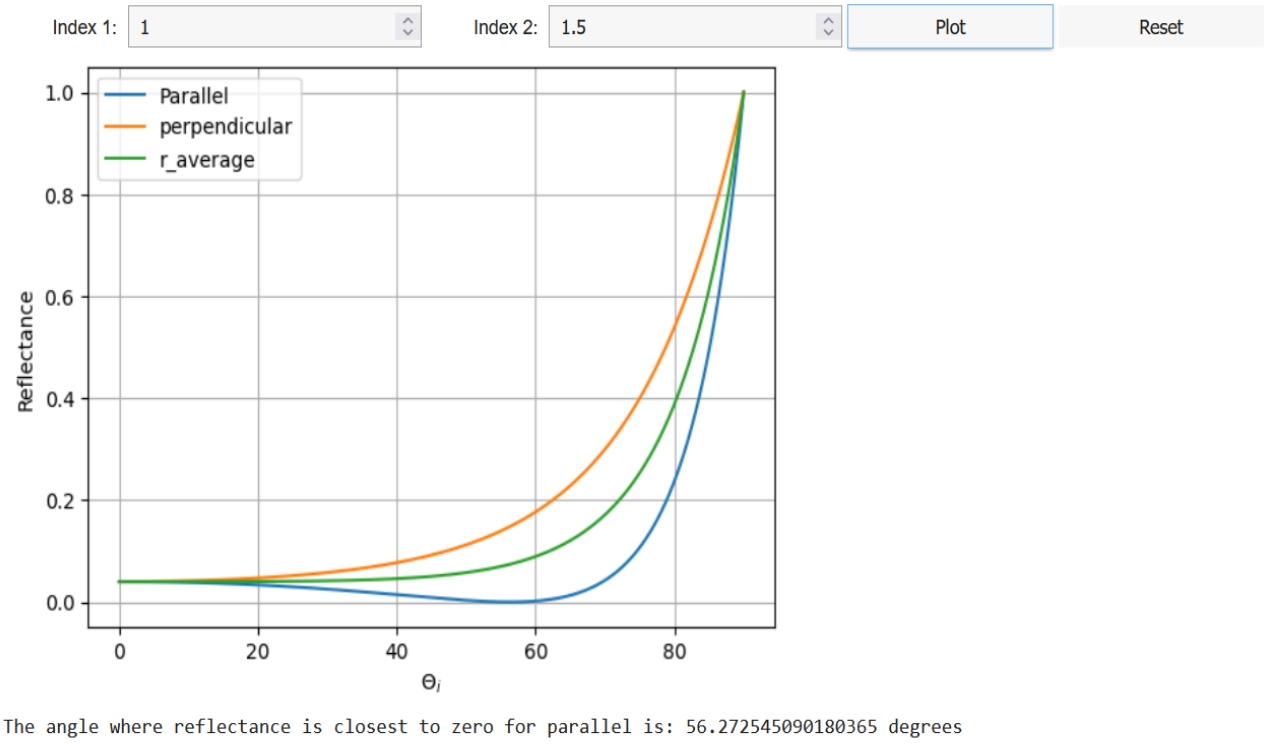


Figure 41: Dielectric Interface applet showing results

For vacuum-diamond interface, where the refractive indices are 1 (vacuum) and 2.42 (diamond), we get the results shown in Figure 42.

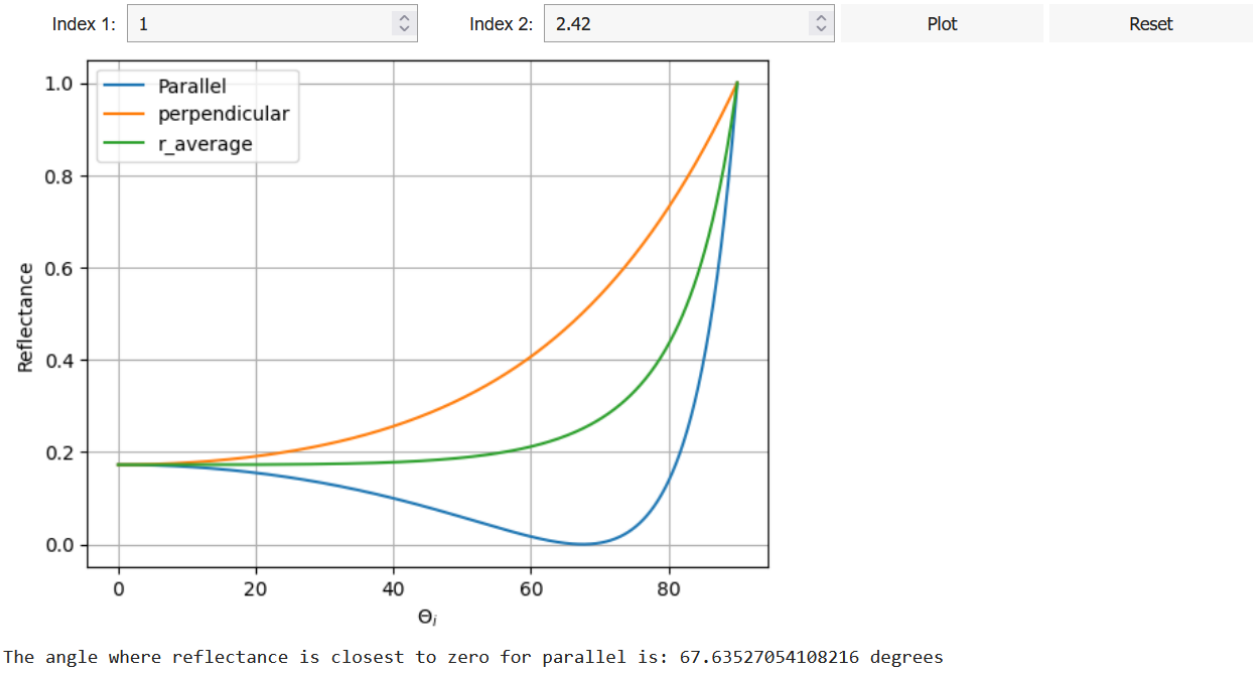


Figure 42:Dielectric Interface Applet with Refractive Indices 1 and 2.4 respectively

For vacuum-diamond interface, zero reflectance was found at 67.64°.

Using an applet provided above offers several benefits. Firstly, it provides an interactive platform where users can input refractive indices of different materials, allowing for hands-on exploration of how light behaves at the interface between these materials. By visualizing the parallel and perpendicular components of the reflected and refracted light rays, users can gain a deeper understanding of how these components change with varying angles of incidence and refractive indices.

Additionally, the applet calculates and displays the average reflectance and transmittance values, providing quantitative insights into the efficiency of light transmission and reflection at the interface. This helps users assess the optical properties of the interface and optimize the design of optical components such as anti-reflection coatings or waveguiding structures for applications in photovoltaic cells or other optical devices.

Moreover, the applet highlights the value of Brewster's angle, a critical angle at which reflected light becomes completely polarized. By identifying Brewster's angle, users can comprehend the

polarization behavior of light at the interface and its implications for minimizing reflection or enhancing specific optical properties.

Overall, this interactive applet serves as a valuable educational tool, enabling users to intuitively grasp complex concepts related to reflection, refraction, and polarization at dielectric interfaces, while also providing practical insights for optical systems with the focus of improving performance and efficiency.

4.7.3 Reflectance and Transmittance

Transmittance and reflectance are crucial factors in optimizing solar panels to generate more radiation from sunlight. Transmittance means allowing sunlight to pass through the panel to reach the solar cells inside, while reflectance refers to minimizing the amount of sunlight that bounces off the panel's surface. By using special coatings and structures like antireflective coatings and textured surfaces, more sunlight gets absorbed by the solar cells, boosting their efficiency. These techniques help in spectral tuning, where the panel's response is tailored to different wavelengths of light for maximum energy conversion. Additionally, advanced designs like multi-junction solar cells benefit from optimizing transmittance and reflectance at different layers to efficiently capture sunlight across a range of wavelengths. Ultimately, these strategies improve the overall performance and output of solar panels, making them a more effective and sustainable source of energy.

The overall average reflectance and average transmittance of a single glazing is given by:

$$\overline{\mathrm{T}} = \propto \left\{ \frac{1}{2}\left[\left(\frac{1-\mathrm{r}_{\parallel}}{1+\mathrm{r}_{\parallel}}\right) + \left(\frac{1-\mathrm{r}_{\perp}}{1+\mathrm{r}_{\perp}}\right)\right]\right\} \qquad (4.7.4)$$

Where $\propto$ is the bulk transmissivity or the property of a material to transmit solar radiation. The overall average reflectance is given by:

$$\overline{\mathrm{R}} = 1 - \overline{\mathrm{T}} \qquad [25] \qquad (4.7.5)$$

A computer program for plotting overall transmittance and reflectance was done (appendix 7 on pages 178-180) and its operational principles are shown in Figures 42 and 43.

For air-glass interface, refractive indices are 1 and 1.5. The results are shown in Figure 42.

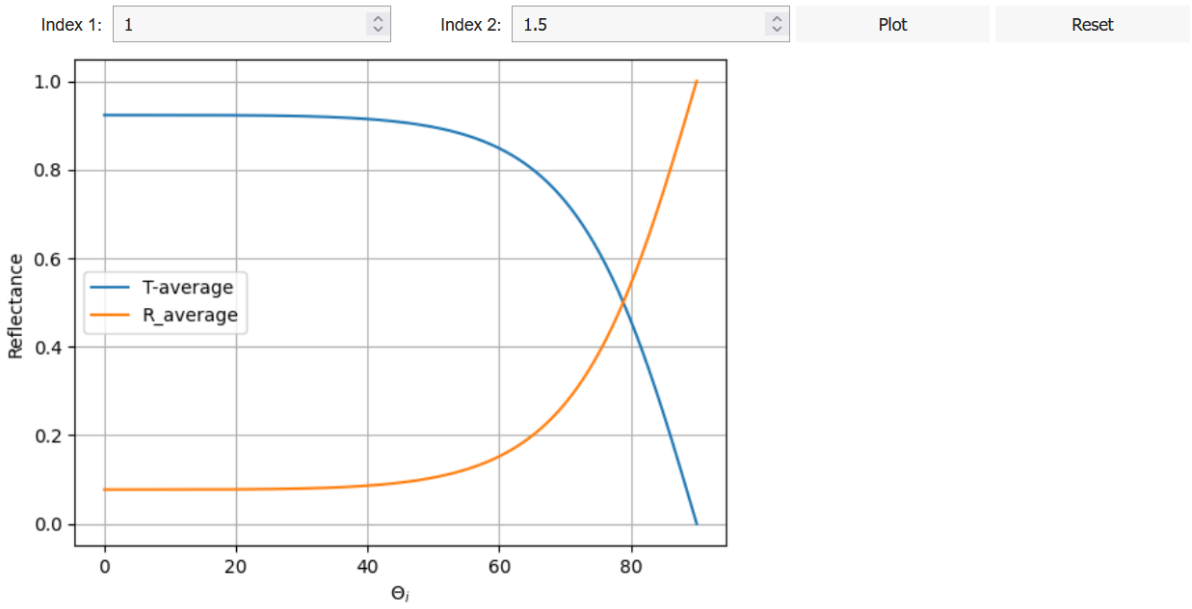


Figure 43: Overall reflectance and transmittance for refractive indices 1 and 1.5

For vacuum-diamond interface, where the refractive indices are 1 and 2.42 and assuming that $\propto$ is equal to 1. The plots shown in Figure 43 were obtained.

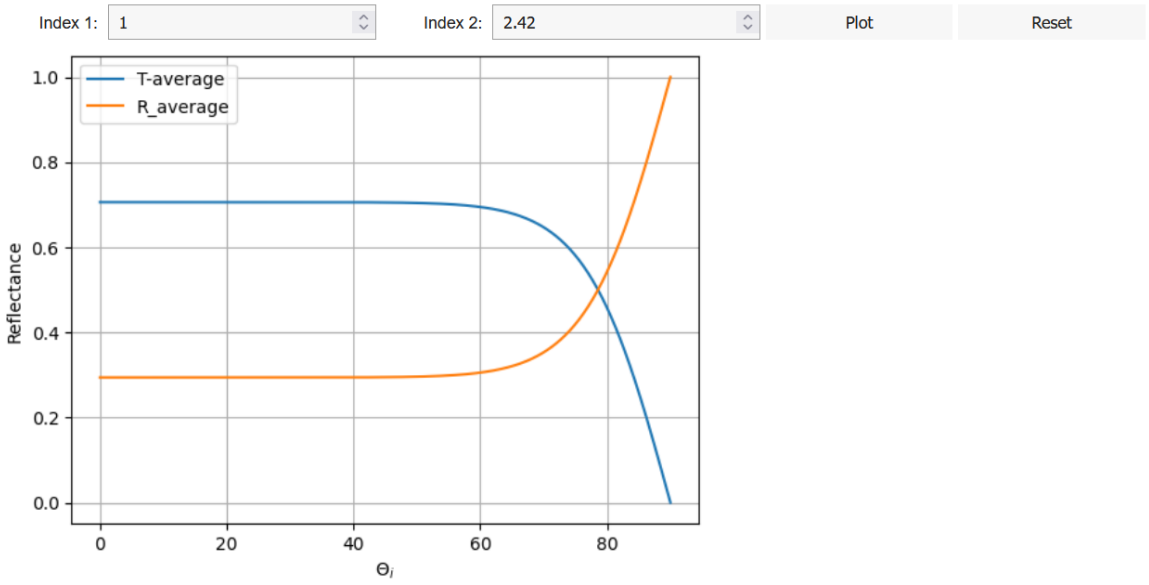


Figure 44: Overall reflectance and transmittance for refractive indices 1 and 2.42

There are many advantages of using the above applet. Firstly, it gives users a fun and interactive way to see how changing the refractive indices of materials affects how light moves through

photovoltaic cells. This helps them understand better because they can see firsthand how these changes impact the solar cell's performance. Secondly, the applet can calculate and show average transmittance and reflectance values at different angles, allowing users to study the cell's optical behavior thoroughly. By looking at these graphs, users can figure out the relationship between reflectance and transmittance which shows that when one goes up, the other goes down. Users can try out different combinations of refractive indices to see how they affect light transmission and reflection, giving them a better understanding of how optics work in photovoltaic technology. Additionally, the applet is engaging because users can change settings and see instant changes in the plotted data. This makes learning more fun and helps users grasp tricky concepts about optics and solar cell performance more easily.

4.7.4 Discussion

In developing the applet, Fresnel's equations for reflection and refraction were utilized, considering two polarized states. These equations, along with the average reflection coefficient equation formed the basis for code implementation. For plotting the variation of the overall transmittance and reflectance of a single glazing an equation for average transmittance was used.

The functionality of the applet is designed to be user-friendly. The user is prompted to input the incident refractive index as index 1, and the refractive index for the second material as index 2. Upon entering these values, the user can generate plots by simply clicking the "plot" button. To reset the output and prepare the applet for the next input, the user can easily click the "reset" button. This approach enhances the user experience, making the applet efficient and accessible for different inputs and scenarios.

4.7.5 Conclusion

In conclusion, the applets developed provides is good for the demonstration of fundamental principles in optics that significantly affect the efficiency of solar energy devices. By simulating the behavior of solar radiation at dielectric interfaces, particularly focusing on the influence of refractive indices and Brewster’s angle, the applet can be a useful tool in the design and optimization of solar panels and concentrators.

Apart from illustrating established concepts such as the relationship between refractive index and reflection/transmission rates, users can interactively explore scenarios involving different

materials and angles of incidence, gaining insights into how these factors affect harnessing of solar energy. By fostering a deeper understanding and inspiring innovative solutions, the applet can contribute to the ongoing evolution of sustainable energy technologies towards a more efficient and environmentally friendly future.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

The creation of applets in solar energy physics is vital for understanding fundamental principles. In this study the following areas were considered: blackbody radiation curves, blackbody fractions, eccentricity correction factor, declination angles, daylengths, extraterrestrial radiation, and reflectance and transmittance at dielectric interfaces. The interactive nature of the applets not only foster a dynamic learning experience but also allow users to manipulate real-time data, providing a personalized and flexible approach to understanding difficult concepts.

The applets demonstrate a clear potential for facilitating research, teaching, and learning in solar energy physics. Their user-friendly interface allows users to customize data inputs, encouraging exploration and experimentation.

To enhance accessibility and usability the applets were converted into executable files that can be installed as standalone applications. It is hoped that these applications would particularly benefit organizations or individuals involved in solar energy, weather, and climate change research, providing a smooth integration into their workflow.

In conclusion, the applets can prove to be a valuable pedagogical and research tool for solar energy physics for teaching and research.

5.2 Recommendations

I recommend that the applets developed be integrated into the physics and engineering curricula covering solar energy. Additionally, I suggest that further development be carried out in order to turn the applets into web applications that can be hosted on the University of Zambia website and accessed online by students, lecturers, researchers and others using various devices such as laptops, tablets and phones.

References

[1] United Nations, "Facts and Figures," United Nations, [Online]. Available: https://www.un.org/en/actnow/facts-and-figures#. [Accessed 8 January 2022].

[2] United Nations, "United Nations Climate Change," United Nations, 12 December 2015. [Online]. Available: https://unfccc.int/process-and-meetings/the-paris-agreement. [Accessed 5 June 2024].

[3] Colorado University, "PhET Interactive Simulations," Colorado University, [Online]. Available: ttps://phet.colorado.edu/en/simulations/blackbody-spectrum. [Accessed 15 June 2022].

[4] C. Woofgang, "Thermal Radiation Curve Applet," Skyserver SDSS, [Online]. Available: https://skyserver.sdss.org/dr1/en/proj/advanced/color/physlet/blackbody.asp. [Accessed 15 June 2022].

[5] Spectralcalc, "Spectral Calc.com," Spectralcalc, [Online]. Available: https://spectralcalc.com/blackbody_calculator/blackbody.php. [Accessed 1 December 2023].

[6] F.-K. Hwang, "Blackbody Radiation Spectrum," 2020. [Online]. Available: https://iwant2study.org/lookangejss/04waves_13electromagneticspectrum/ejss_model_BlackbodyRadiationSpectrumwee. [Accessed 7 June 2024].

[7] UNL Astronomy , "Daylight Hours Explorer," UNL Astronomy, [Online]. Available: https://astro.unl.edu/classactio/animation/coordsmotion/daylighthoursexplorer.html. [Accessed 17 June 2022].

[8] timeanddate.com, "Sun Calculator," Time and Date AS, [Online]. Available: https://www.timeanddate.com/sun/. [Accessed 5 January 2024].

[9] soda-pro.com, "Extraterrestrial Irradiance and Top of Atmosphere," soda-pro.com, [Online]. Available: https://www.soda-pro.com/web-services/radiation/extraterrestrial-irradiance-and-toa. . [Accessed 17 June 2022].

[10] Santa Clara University, "Calculation of Extraterrestrial Solar Radiation," Santa Clara University, [Online]. Available: https://www.engr.scu.edu/~emaurer/tools/calc_solar_cgi.pl. [Accessed 16 December 2023].

[11] Laser Calculator, "Fresnel reflection and transmission calculator," Lasercalculator, [Online]. Available: https://www.lasercalculator.com/fresnel-reflection-and-transmission-calculator/. [Accessed 28 June 2022].

[12] M. Iqbal, An Introduction to Solar Radiation, Ontario: Academic Press Canada, 1983.

[13] Python, "Python 3.9.18 documentation," Python, [Online]. Available: Available: https://docs.python.org/3.9/. [Accessed 12 July 2021].

[14] geeksforgeeks, "History of Python," geeksforgeeks, [Online]. Available: https://www.geeksforgeeks.org/history-of-python/. [Accessed 10 January 2022].

[15] Project Jupyter Documentation, "Jupyter," Jupyter, [Online]. Available: https://docs.jupyter.org/en/latest/. [Accessed 20 February 20].

[16] R. Johansson, Numerical Python: Scientific Computing and Data Science Applications with Numpy, Scipy and Matplotlib, Urayasu-shi, Chiba: Apress, 2018.

[17] G. Moruzzi, Essential Python for the Physicist, Cham: Springer Nature Switzerland, 2020.

[18] J. Hunter, "Matplotlib: A 2D Graphics Environment," *IEEE Xplore,* vol. 9, no. 3, pp. 90-95, 2007.

[19] Pandas, "Pandas," AQR Capital Management, [Online]. Available: https://pandas.pydata.org. [Accessed 25 May 2022].

[20] J. Widgets, "ipywidgets," Jupyter Widgets, [Online]. Available: https://ipywidgets.readthedocs.io. [Accessed 13 July 2023].

[21] P. S. F. L. v. 2, "tkinter — Python interface to Tcl/Tk," 2001-2024. [Online]. Available: https://docs.python.org/3/library/tkinter.html. [Accessed 26 March 2024].

[22] The Editors of Encyclopaedia, "Britannica," Encyclopaedia Britannica, [Online]. Available: https://www.britannica.com/biography/Josef-Stefan. [Accessed 24 March 2022].

[23] The Editors of Encyclopaedia, "Wilhem Wien," Encyclopaedia Britannica, [Online]. Available: https://www.britannica.com/biography/Wilhelm-Wien. [Accessed 24 March 2022].

[24] The Editors of Encyclopaedia, "Planck's Radiation Law," Encyclopaedia Britannica, [Online]. Available: https://www.britannica.com/science/Plancks-radiation-law.. [Accessed 27 March 2022].

[25] S. Wieder, An Introduction to Solar Energy for Scientists and Engineers, Wiley, 1982.

[26] C3.ai, "Glossary: Root Mean Square Root Error (RMSE)," C3.ai, [Online]. Available: https://c3.ai/glossary/data-science/root-mean-square-error-rmse/. [Accessed 23 December 2023].

[27] D. Lawson, "Blackody Fraction, Infinite Series and Spreadsheets," *International Journal of Engineering Education,* vol. 20, no. 6, pp. 984-990, 2004.

[28] K. A. Stroud and D. J. Booth, Engineering Mathematics, Seventh ed., London: Palgrave Macmillan, 2013.

[29] M. Perino, "Wikepedia," IDES-EDU, Intelligent Energy Europe, [Online]. Available: https://docplayer.net/12611245-Lecture-n-3-solar-energy-and-solar-radiation-ides-edu.html. [Accessed 6 January 2024].

[30] A. Karafil, H. Özbay, M. Kesler and H. Parmaksiz, "ResearchGate," Bilecik Seyh Edebali University, [Online]. Available: : https://www.researchgate.net/publication/298318962.. [Accessed 20 March 2024].

[31] Collins, "Collins: Language ,daylength," Collins Dictionary, [Online]. Available: https://www.collinsdictionary.com/dictionary/english/daylength. [Accessed 17 August 2022].

[32] Sandia National Laboratories, "Sandia National Laboratories," Research, Earth, Energy and Environmental Science, [Online]. Available: https://pvmc.sandia.gov/modeling-guide/1-weather-design-inputs/irradianceinsolation/extraterrestrial-radiation/. [Accessed 12 August 2023].

[33] Y.-C. Hsiao, H. Zang, I. Ivanov, T. Xu, L. Lu, L. Yu and B. Hu, "Dielectric Interface Effects on Surface Charge Accumulation and Collection towards High-Efficiency Organic Solar Cells," *Journal of Applied Physics,* vol. 115, no. 15, 2014.

[34] J. R. Howell, M. M. Pinar and R. Siegel, Thermal Radiation Heat Transfer, Florida: CRC Press, 2016.

[35] B. E. Saleh and M. C. Teich, Fundamentals of Photonics, Hoboken, NJ: Wiley, 1991.

[36] Wikepedia, "Stefan-Bolzmann's law," Wikepedia, [Online]. Available: https://en.wikipedia.org/wiki/Stefan%E2%80%93Boltzmann_law. [Accessed 3 January 2023].

[37] P. C. Jain, ""Identification of dissertation problems," Written notes to the author," Lusaka, 2021.

Appendices

Appendix 1: Computer Codes for Blackbody Radiation Curves

```
# Creating the function for calculating wavelengths and power density
def spectral_power_density(T):
 """Returns the radiant power per unit area of blackbody at absolute   temperature T in wavelength range 900nm-Kelvin to 2 microns-Kelvin"""
 # import numpy using alias np
 import numpy as np
 from numpy import pi
 # Create the physical constants
 h=6.6252e-34  # Planck's constant
 k=1.3806e-23 # Boltzmann's constant
 c=2.9979e8 # Speed of light in a vacuum
 # Define constant a = 2 pi * h * c**2
 a= 2*pi*h*c**2
 # Define constant b = h * c / k
 b = h*c/k
#Create an array of 10000 linearly spaced values ranging from 900nm to 2μm
# and store it in variable wavelength
 #wavelength = np.linspace(0.01e-6,2e-6,10000)
 if T <= 2000:
   wavelength = np.linspace(9e-7,700e-7,10000)
 else:
   wavelength = np.linspace(0.01e-6,2e-6,10000)
```

```
# Calculate blackbody radiant power: radiant_power
radiant_power = a/ (np.power(wavelength,5)*(np.exp(b/(wavelength*T))-1))
# return wavelength and radiant power as tuple
return(wavelength, radiant_power)
```

The blackbody radiant power for absolute temperature, T=288 K was calculated by calling the function created above.

```
wavelength_288K,radiant_power_288K = spectral_power_density(T=288)
```

For T=5 000 K

```
wavelength_5000K,radiant_power_5000K = spectral_power_density(T = 5000)
```

The blackbody curve at absolute temperature, T=288 K was plotted using code shown below.

```
#Import data visualization package matplotlib.pyplot using alias plt
import matplotlib.pyplot as plt

#Initiating a new plot
plt.figure()
#Generating the plot for the blackbody curve, where the wavelength_288K represents the x- axis
#while the radiant_power_288K represents the y-axis. The curve is labeled as "288 K" and the
#assigned color is black (color="k").
plt.plot(wavelength_288K, radiant_power_288K,label="288 K", color="k")
#Setting the x-axis label
plt.xlabel("$\lambda(m)$")
#Setting the y-axis label
plt.ylabel("$B_\lambda(T)~~W/m^{2}$")
#Insert the grid to the plot
```

```
plt.grid()
#Adding the legend to the plot
plt.legend()
#Saving the plot with a filename in jpg format
plt.savefig("Black body Curve for 288K.jpg")
#This line of code displays or shows the figure
plt.show()
```

The absolute temperatures of 5000K, 7000K, 8000K, 9000K and 10000K were plotted on the same graph using the following code.

```
plt.figure()
plt.plot(wavelength_5000K, radiant_power_5000K,label="5000 K", color="r")
plt.plot(wavelength_5775K, radiant_power_5775K,label="5775 K", color="y")
plt.plot(wavelength_7000K, radiant_power_7000K,label="7000 K", color="g")
plt.plot(wavelength_8000K, radiant_power_8000K,label="8000 K", color="lightblue")
plt.plot(wavelength_9000K, radiant_power_9000K,label="9000 K", color="b")
plt.plot(wavelength_10000K, radiant_power_10000K,label="10000 K", color="darkblue")
plt.xlabel("$\lambda(m)$")
plt.ylabel("$B_\lambda(T)~~W/m^{2}$")
plt.grid()
plt.legend()
plt.savefig("Black body Curve for 5 775K,7000, 8000, 9000, 10000.jpg")
plt.show()
```

The following computer program was used to find the peak wavelengths.

```
# Calling the function
wavelength_5000K, radiant_power_5000K = spectral_power_density(T = 5000)
#Creating a Boolean filter to get the maximum value of the radiant power density
filter_5000K = radiant_power_5000K == np.max(radiant_power_5000K)
#This line of code retrieves the peak wavelength from the maximum value of the radiant power
planck_peak_wavelength_5000K = wavelength_5000K[filter_5000K]
#Print the result [20]
print(planck_peak_wavelength_5000K)
```

Finding peak wavelengths for different temperatures, the following code was used

```
# Create an empty list
planck_peak_wavelengths = []
planck_peak_wavelengths_list=[planck_peak_wavelength_5000K,
                              planck_peak_wavelength_5775K,
                              planck_peak_wavelength_7000K,
                              planck_peak_wavelength_8000K,
                              planck_peak_wavelength_9000K,
                              planck_peak_wavelength_10000K]
# Iterating each element using a loop
for i in planck_peak_wavelengths_list :
# Append the elements of planck_peak_wavelengths_list to #planck_peak_wavelength
    planck_peak_wavelengths.append(i)
# ravel column matrix into row vector
planck_peak_wavelengths = np.array(planck_peak_wavelengths).ravel()
# print out row vector
```

```
print(planck_peak_wavelengths)
```

Code for plotting peak wavelengths at different temperatures. This resulted in an output shown in Figure 4.

```
plt.figure()

plt.scatter(temps, planck_peak_wavelengths)

plt.ylabel("Peak Wavelength [m]")

plt.xlabel("Blackbody temperature [K]")

plt.grid()

plt.savefig("Blackbody Planck Peak Wavelength.jpg")

plt.show()
```

Using equation (4.1.6), the following computer program was created to calculate the peak wavelength.

```
def wien_displacement_law(temp):
 """Returns the peak wavelength of a blackbody at a specific temperature."""
 # set proportionality constant: alpha
 alpha = 2898e-6 # in units of m.K
 # calculate the peak wavelength : peak_wavelength in meters
 peak_wavelength = alpha/ temp
 # return peak wavelength: peak_wavelength
 return peak_wavelength
```

The created Wien's displacement law function was called upon to calculate the maximum wavelengths at different temperatures as follows:

For Temperature, T=5 000K.

```
max_wavelength_5000K = wien_displacement_law(temp = 5000)

print(max_wavelength_5000K)
```

For Temperature, *T*=5 775K.

```
max_wavelength_5775K = wien_displacement_law(temp = 5775)

print(max_wavelength_5775K)

wien_peak_wavelengths = []

wien_peak_wavelengths_list = [max_wavelength_5000K,max_wavelength_5775K,

                    max_wavelength_7000K,max_wavelength_8000K,

                   max_wavelength_9000K,max_wavelength_10000K]

for i in wien_peak_wavelengths_list :

  wien_peak_wavelengths.append(i)

# ravel column matrix into row vector

wien_peak_wavelengths = np.array(wien_peak_wavelengths).ravel()

# print out row vector

print(wien_peak_wavelengths)
```

Code for plotting peak wavelengths from Wien's and Planck's laws

```
plt.figure()

plt.plot(temps, wein_peak_wavelengths,"r.")

plt.scatter(temps, planck_peak_wavelengths)

plt.ylabel("Peak Wavelength [m]")

plt.xlabel("Blackbody Temperature [K]")

plt.grid()

plt.legend(['Wein', 'Planck'])

plt.savefig("Blackbody Peak Wavelength as function of temperature.jpg")

plt.show()
```

The following code was used to calculate the root mean square error.

```
def root_mean_square_error(wein_peak_wavelengths, \ planck_peak_wavelengths):
    # We find the squared differences of the corresponding values
    squared_differences = np.square(planck_peak_wavelengths – wein_peak_wavelengths)
    # We get the mean squared error from the squared differences
    mean_squared_error = np.mean(squared_differences)
    # Calculating the root mean squared error
    root_mean_squared_error = np.sqrt(mean_squared_error)
    return root_mean_squared_error
Solution = root_mean_square_error(wein_peak_wavelengths, planck_peak_wavelengths)
print("Root Mean Square Error:", Solution)
```

Code for calculating the average percentage error.

```
def calculate_percentage_error(wein_peak_wavelengths, planck_peak_wavelengths):
    # Calculate percentage error for each pair of values
    errors = [(abs(planck_peak_wavelengths[i] - wein_peak_wavelengths[i])/
            planck_peak_wavelengths[i]) * 100 for i in range(len(planck_peak_wavelengths))]
    # Calculate the average percentage error
    average_error = sum(errors) / len(errors)
    return average_error
percentage_error = calculate_percentage_error(wein_peak_wavelengths,
                                    planck_peak_wavelengths)
print(f"Average percentage error: {percentage_error:.2f}%")
```

Calculating the Blackbody radiant power using the Stefan-Boltzmann's law

```
def stefan_boltzmann_law(temp):
 """Return the power of a black body at temperature T"""
 # define constant sigma
 sigma = 5.6696e-8
 # calculating the blackbody radiant power
 radiant_power = sigma*np.power(temp,4)
 # return the radiant power
 return radiant_power
```

Calling the stefan_boltzmann_law() function to calculate the blackbody radiant power at temperature of T=5000*K*

```
stefan_radiant_power_5000K = stefan_boltzmann_law(temp= 5000)
print(stefan_radiant_power_5000K)
```

At temperature of T=5775K

```
stefan_radiant_power_5775K = stefan_boltzmann_law(temp= 5775)
print(stefan_radiant_power_5775K)
```

The graph of Stefan radiant power at different temperatures was plotted using the code shown below.

```
# create list from calculated stefan radiant power at different temperature
stefan_radiant_powers_list = [stefan_radiant_power_5000K,stefan_radiant_power_5775K,
                              stefan_radiant_power_7000K,stefan_radiant_power_8000K,
                              stefan_radiant_power_9000K,stefan_radiant_power_10000K]
# create numpy array
stefan_radiant_powers = np.array(stefan_radiant_powers_list)
plt.figure()
```

```
plt.scatter(temps, stefan_radiant_powers)
plt.title(" Stefan Blackbody Radiant Power")
plt.ylabel("Radiant Power [W]")
plt.xlabel("Blackbody temp [K]")
plt.grid()
plt.savefig("Stefan Blackbody radiant power as function of temperature.jpg")
plt.show()
```

Calculating radiant power using Simpson's numerical integration

```
from scipy.integrate import simpson
```

Calling on the simpson() function the radiant power of a blackbody at temperature of T=5000K was calculated using the code shown below.

```
planck_radiant_power_5000K = simpson(y = radiant_power_5000K, x = wavelength_5000K)
print(planck_radiant_power_5000K)
```

The radiant power of blackbody at temperature of T=5775K was calculated by calling the simpson() function.

```
planck_radiant_power_5775K = simpson(y = radiant_power_5775K, x = wavelength_5775K)
print(planck_radiant_power_5775K)
```

The radiant power of a blackbody was calculated at temperatures 7000K, 8000K, 9000K and 1000K and plotted using the code shown below.

```
# create list from calculated stefan radiant powers
simpson_radiant_powers_list= [simpson_radiant_power_5000K,simpson_radiant_power_5775K,
                              simpson_radiant_power_7000K,simpson_radiant_power_8000K,
                              simpson_radiant_power_9000K,simpson_radiant_power_10000K]
# create numpy array
simpson_radiant_powers = np.array(simpson_radiant_powers_list)
```

```
plt.figure()
plt.scatter(temps, simpson_radiant_powers)
plt.title("Planck Blackbody Radiant Power")
plt.ylabel("Radiant Power [W]")
plt.xlabel("Blackbody temp [K]")
plt.grid()
plt.savefig("simpson Blackbody radiant power as function of temperature.jpg")
plt.show()
```

A code for plotting radiant power using Stefan and Simpson methods is shown below

```
plt.figure()
plt.scatter(temps, stefan_radiant_powers)
plt.scatter(temps, simpson_radiant_powers)
plt.title("Blackbody Radiant Power")
plt.ylabel("Radiant Power [W]")
plt.xlabel("Blackbody Temperature [K]")
plt.grid()
plt.legend(['Stefan', 'Planck'])
plt.savefig("Blackbody radiant power as function of temperature.jpg")
plt.show()
```

Calculating the average percentage error the following code was used

```
def calculate_percentage_error(simpson_radiant_powers,  stefan_radiant_powers):
   # Calculate percentage error for each pair of values
        errors = [(abs(stefan_radiant_powers[i] - simpson_radiant_powers[i])/
        stefan_radiant_powers[i])  * 100 for i in range(len(stefan_radiant_powers))]
```

```
    # Calculate the average percentage error
    average_error = sum(errors) / len(errors)
    return average_error
percentage_error = calculate_percentage_error(stefan_radiant_powers,  simpson_radiant_powers)
print(f"Average Percentage Error: {percentage_error:.2f}%")
```

Appendix 2: Computer Codes and Tables of Blackbody Fractions

The following computer code was used to calculate blackbody fractions.

```
# Import packages

import numpy as np

from numpy import pi

import matplotlib.pyplot as plt

import pandas as pd

# define Planck's constant

h = 6.6252e-34

# define Boltzmann's constant

k = 1.3806e-23

# define speed of light in vaccum

c = 2.9979e8

# define b

b = h*c*1e6/k

# Initialize an empty list for storing calculated values of fractional radiance

calculated_fractional_radiance = []

# Loop through each value in the list

x = np.arange(1100,50100,100)

for i in x :

    # Initialize a variable to store the sum

    sum_x = 0

    # Loop through the range from 1 to 20

    for n in range(1,21):
```

```
        # calculate the fractional radiance
        ans = np.sum((15/(pi**4))*(((b**3)/(n*i**3)) +((3*b**2)/((n**2)*
            (i**2)))+((6*b)/(i*n**3)+(6/n**4)))*(np.exp(-((n*b)/i))))
        # Add the calculated term to the sum_x variable
        sum_x = sum_x + ans
    # Append the sum of fractional radiance terms to the list
    calculated_fractional_radiance.append(sum_x)
# Create a DataFrame from the provided data
df = pd.DataFrame({'x': x, 'f(x)': calculated_fractional_radiance})
# Split the DataFrame into six sets
df_set1 = df.loc[:81].reset_index(drop=True)
df_set2 = df.loc[82:163].reset_index(drop=True)
df_set3 = df.loc[164:245].reset_index(drop=True)
df_set4 = df.loc[246:327].reset_index(drop=True)
df_set5 = df.loc[328:409].reset_index(drop=True)
df_set6 = df.loc[410:489].reset_index(drop=True)
# Display the six sets next to each other vertically
result_df = pd.concat([df_set1, df_set2, df_set3, df_set4, df_set5, df_set6], axis=1)
# Rename the columns for clarity
result_df.columns = ['x', 'f(x)', 'x', 'f(x)', 'x', 'f(x)','x', 'f(x)', 'x', 'f(x)', 'x', 'f(x)']
# Save the concatenated DataFrame to an excel file
result_df.to_excel('output_calculated_radiance.xlsx', index=False)
# Display the concatenated DataFrame in a table format without the index column
print(result_df)
```

```
result_df
```

For plotting the graph of blackbody energy fractions the following code was employed.

```
# Plot of calculated fractional radiance curve

# create a plot figure

plt.figure()

# add title "calculated fractional radiance"

plt.title("calculated fractional radiance")

# make line plot of calculated fractional radiance

plt.plot(x, calculated_fractional_radiance, label='calculated fractional radiance')

# add xlabel

plt.xlabel("$x~( m\cdot K)$")

# add y label

plt.ylabel('$f_{\mathrm{cal}}(x)$')

# add grid

plt.grid()

# add legend

plt.legend()

# save plot figure as "tabulated_spectral_radiance.jpg"

plt.savefig("calculated_spectral_radiance.jpg")

# show plot

plt.show()
```

Table of Calculated Blackbody Fractions

x	f(x)	x	f(x)	x	f(x)	x	f(x)	x	f(x)	x	f(x)
1100	0.000912	9300	0.898102	17500	0.979318	25700	0.992746	33900	0.996666	42100	0.998201
1200	0.002136	9400	0.900625	17600	0.97963	25800	0.992824	34000	0.996694	42200	0.998213
1300	0.00432	9500	0.903068	17700	0.979936	25900	0.9929	34100	0.996721	42300	0.998226
1400	0.007797	9600	0.905433	17800	0.980236	26000	0.992976	34200	0.996748	42400	0.998237
1500	0.012859	9700	0.907724	17900	0.98053	26100	0.993051	34300	0.996775	42500	0.998249
1600	0.019732	9800	0.909944	18000	0.980818	26200	0.993124	34400	0.996801	42600	0.998261
1700	0.028552	9900	0.912094	18100	0.9811	26300	0.993197	34500	0.996828	42700	0.998273
1800	0.039364	10000	0.914178	18200	0.981377	26400	0.993268	34600	0.996853	42800	0.998284
1900	0.052135	10100	0.916198	18300	0.981649	26500	0.993339	34700	0.996879	42900	0.998296
2000	0.066762	10200	0.918157	18400	0.981915	26600	0.993408	34800	0.996905	43000	0.998307
2100	0.083091	10300	0.920056	18500	0.982176	26700	0.993477	34900	0.99693	43100	0.998318
2200	0.100932	10400	0.921899	18600	0.982433	26800	0.993544	35000	0.996954	43200	0.998329

2300	0.120077	10500	0.923686	18700	0.982684	26900	0.993611	35100	0.996979	43300	0.99834
2400	0.140309	10600	0.92542	18800	0.982931	27000	0.993677	35200	0.997003	43400	0.998351
2500	0.161412	10700	0.927103	18900	0.983173	27100	0.993742	35300	0.997027	43500	0.998362
2600	0.18318	10800	0.928737	19000	0.98341	27200	0.993806	35400	0.997051	43600	0.998373
2700	0.20542	10900	0.930323	19100	0.983643	27300	0.993869	35500	0.997075	43700	0.998384
2800	0.227956	11000	0.931863	19200	0.983872	27400	0.993931	35600	0.997098	43800	0.998394
2900	0.250629	11100	0.93336	19300	0.984097	27500	0.993993	35700	0.997121	43900	0.998405
3000	0.2733	11200	0.934813	19400	0.984317	27600	0.994053	35800	0.997144	44000	0.998415
3100	0.295848	11300	0.936225	19500	0.984533	27700	0.994113	35900	0.997166	44100	0.998425
3200	0.318171	11400	0.937597	19600	0.984746	27800	0.994172	36000	0.997189	44200	0.998435
3300	0.34018	11500	0.938931	19700	0.984954	27900	0.99423	36100	0.997211	44300	0.998446
3400	0.361805	11600	0.940228	19800	0.985159	28000	0.994288	36200	0.997233	44400	0.998456
3500	0.382985	11700	0.941489	19900	0.98536	28100	0.994344	36300	0.997254	44500	0.998465
3600	0.403675	11800	0.942715	20000	0.985558	28200	0.9944	36400	0.997276	44600	0.998475

3700	0.423839	11900	0.943907	20100	0.985752	28300	0.994455	36500	0.997297	44700	0.998485
3800	0.443449	12000	0.945068	20200	0.985942	28400	0.99451	36600	0.997318	44800	0.998495
3900	0.462487	12100	0.946196	20300	0.98613	28500	0.994564	36700	0.997338	44900	0.998504
4000	0.48094	12200	0.947295	20400	0.986314	28600	0.994617	36800	0.997359	45000	0.998514
4100	0.498803	12300	0.948364	20500	0.986494	28700	0.994669	36900	0.997379	45100	0.998523
4200	0.516074	12400	0.949405	20600	0.986672	28800	0.994721	37000	0.997399	45200	0.998533
4300	0.532755	12500	0.950419	20700	0.986846	28900	0.994772	37100	0.997419	45300	0.998542
4400	0.548852	12600	0.951406	20800	0.987018	29000	0.994822	37200	0.997439	45400	0.998551
4500	0.564375	12700	0.952367	20900	0.987186	29100	0.994872	37300	0.997459	45500	0.99856
4600	0.579333	12800	0.953304	21000	0.987352	29200	0.994921	37400	0.997478	45600	0.998569
4700	0.593739	12900	0.954216	21100	0.987514	29300	0.99497	37500	0.997497	45700	0.998578
4800	0.607608	13000	0.955105	21200	0.987674	29400	0.995018	37600	0.997516	45800	0.998587
4900	0.620953	13100	0.955972	21300	0.987832	29500	0.995065	37700	0.997535	45900	0.998596
5000	0.633791	13200	0.956816	21400	0.987986	29600	0.995111	37800	0.997553	46000	0.998605

5100	0.646138	13300	0.95764	21500	0.988138	29700	0.995158	37900	0.997571	46100	0.998613
5200	0.65801	13400	0.958443	21600	0.988288	29800	0.995203	38000	0.99759	46200	0.998622
5300	0.669424	13500	0.959225	21700	0.988435	29900	0.995248	38100	0.997608	46300	0.99863
5400	0.680396	13600	0.959989	21800	0.988579	30000	0.995292	38200	0.997625	46400	0.998639
5500	0.690942	13700	0.960734	21900	0.988721	30100	0.995336	38300	0.997643	46500	0.998647
5600	0.701079	13800	0.96146	22000	0.988861	30200	0.99538	38400	0.997661	46600	0.998656
5700	0.710822	13900	0.962169	22100	0.988999	30300	0.995422	38500	0.997678	46700	0.998664
5800	0.720187	14000	0.962861	22200	0.989134	30400	0.995465	38600	0.997695	46800	0.998672
5900	0.729189	14100	0.963537	22300	0.989267	30500	0.995506	38700	0.997712	46900	0.99868
6000	0.737842	14200	0.964196	22400	0.989398	30600	0.995547	38800	0.997729	47000	0.998688
6100	0.746162	14300	0.964839	22500	0.989527	30700	0.995588	38900	0.997745	47100	0.998696
6200	0.754161	14400	0.965468	22600	0.989654	30800	0.995628	39000	0.997762	47200	0.998704
6300	0.761853	14500	0.966081	22700	0.989778	30900	0.995668	39100	0.997778	47300	0.998712
6400	0.769251	14600	0.966681	22800	0.989901	31000	0.995707	39200	0.997794	47400	0.99872

6500	0.776367	14700	0.967266	22900	0.990022	31100	0.995746	39300	0.99781	47500	0.998727
6600	0.783213	14800	0.967838	23000	0.990141	31200	0.995784	39400	0.997826	47600	0.998735
6700	0.7898	14900	0.968397	23100	0.990258	31300	0.995822	39500	0.997842	47700	0.998743
6800	0.79614	15000	0.968943	23200	0.990373	31400	0.99586	39600	0.997857	47800	0.99875
6900	0.802242	15100	0.969476	23300	0.990486	31500	0.995897	39700	0.997873	47900	0.998758
7000	0.808117	15200	0.969998	23400	0.990598	31600	0.995933	39800	0.997888	48000	0.998765
7100	0.813774	15300	0.970508	23500	0.990707	31700	0.995969	39900	0.997903	48100	0.998773
7200	0.819223	15400	0.971006	23600	0.990816	31800	0.996005	40000	0.997918	48200	0.99878
7300	0.824472	15500	0.971493	23700	0.990922	31900	0.99604	40100	0.997933	48300	0.998787
7400	0.82953	15600	0.97197	23800	0.991027	32000	0.996075	40200	0.997947	48400	0.998794
7500	0.834404	15700	0.972436	23900	0.99113	32100	0.996109	40300	0.997962	48500	0.998801
7600	0.839103	15800	0.972892	24000	0.991232	32200	0.996143	40400	0.997976	48600	0.998808
7700	0.843633	15900	0.973338	24100	0.991332	32300	0.996177	40500	0.99799	48700	0.998815
7800	0.848003	16000	0.973774	24200	0.99143	32400	0.99621	40600	0.998005	48800	0.998822

7900	0.852217	16100	0.974201	24300	0.991528	32500	0.996243	40700	0.998019	48900	0.998829
8000	0.856284	16200	0.974619	24400	0.991623	32600	0.996276	40800	0.998032	49000	0.998836
8100	0.860208	16300	0.975028	24500	0.991718	32700	0.996308	40900	0.998046	49100	0.998843
8200	0.863997	16400	0.975428	24600	0.99181	32800	0.99634	41000	0.99806	49200	0.99885
8300	0.867654	16500	0.97582	24700	0.991902	32900	0.996371	41100	0.998073	49300	0.998856
8400	0.871186	16600	0.976204	24800	0.991992	33000	0.996402	41200	0.998087	49400	0.998863
8500	0.874598	16700	0.976579	24900	0.992081	33100	0.996433	41300	0.9981	49500	0.99887
8600	0.877895	16800	0.976947	25000	0.992168	33200	0.996463	41400	0.998113	49600	0.998876
8700	0.881081	16900	0.977307	25100	0.992254	33300	0.996493	41500	0.998126	49700	0.998883
8800	0.88416	17000	0.97766	25200	0.992339	33400	0.996523	41600	0.998139	49800	0.998889
8900	0.887137	17100	0.978005	25300	0.992423	33500	0.996552	41700	0.998151	49900	0.998896
9000	0.890016	17200	0.978344	25400	0.992505	33600	0.996581	41800	0.998164	50000	0.998902
9100	0.892801	17300	0.978675	25500	0.992587	33700	0.996609	41900	0.998177		
9200	0.895495	17400	0.979	25600	0.992667	33800	0.996638	42000	0.998189		

Table of Blackbody Fractions (Wieder, 1982, p. 14)

x	f(x)	x	f(x)	x	f(x)
1100	0.001	4600	0.58	8100	0.86
1200	0.002	4700	0.594	8200	0.864
1300	0.004	4800	0.608	8300	0.868
1400	0.008	4900	0.621	8400	0.871
1500	0.013	5000	0.634	8500	0.875
1600	0.02	5100	0.646	8600	0.878
1700	0.029	5200	0.658	8700	0.881
1800	0.04	5300	0.669	8800	0.884
1900	0.052	5400	0.68	8900	0.887
2000	0.067	5500	0.691	9000	0.89
2100	0.083	5600	0.701	9100	0.893
2200	0.101	5700	0.711	9200	0.895
2300	0.12	5800	0.72	9300	0.898
2400	0.14	5900	0.73	9400	0.901
2500	0.161	6000	0.738	9500	0.903
2600	0.183	6100	0.746	9600	0.905
2700	0.205	6200	0.754	9700	0.908

2800	0.228	6300	0.762	9800	0.91
2900	0.251	6400	0.77	9900	0.912
3000	0.273	6500	0.776	10000	0.914
3100	0.296	6600	0.783	11000	0.932
3200	0.318	6700	0.79	12000	0.945
3300	0.34	6800	0.796	13000	0.955
3400	0.362	6900	0.802	14000	0.963
3500	0.383	7000	0.808	15000	0.969
3600	0.404	7100	0.814	16000	0.974
3700	0.424	7200	0.819	17000	0.978
3800	0.443	7300	0.824	18000	0.981
3900	0.462	7400	0.83	19000	0.983
4000	0.483	7500	0.834	20000	0.986
4100	0.499	7600	0.84	30000	0.995
4200	0.516	7700	0.844	40000	0.998
4300	0.533	7800	0.848	50000	0.999
4400	0.549	7900	0.852		
4500	0.564	8000	0.856		

Appendix 3: Tables and Computer Codes for Eccentricity Correction Factor

Eccentricity Correction Factor obtained from Iqbal (Iqbal, 1983)

Date	Jan	Feb	Mar	Apr	May	June	Jul	Aug	Sep	Oct	Nov	Dec
1	1.035	1.0306	1.019	1.0014	0.9845	0.9717	0.9666	0.97	0.9814	0.9976	1.0155	1.0291
2	1.0351	1.0303	1.0185	1.0008	0.984	0.9714	0.9666	0.9703	0.9819	0.9982	1.0161	1.0295
3	1.0351	1.03	1.018	1.0002	0.9835	0.9712	0.9666	0.9705	0.9823	0.9988	1.0166	1.0298
4	1.0351	1.0297	1.0174	0.9997	0.983	0.9709	0.9666	0.9708	0.9828	0.9994	1.0171	1.0301
5	1.0351	1.0294	1.0169	0.9991	0.9825	0.9706	0.9666	0.9711	0.9833	0.9999	1.0177	1.0304
6	1.035	1.029	1.0164	0.9985	0.9821	0.9703	0.9666	0.9713	0.9838	1.0005	1.0182	1.0307
7	1.035	1.0287	1.0158	0.9979	0.9816	0.9701	0.9666	0.9716	0.9843	1.0011	1.0187	1.031
8	1.035	1.0283	1.0153	0.9973	0.9811	0.9698	0.9666	0.9719	0.9848	1.0017	1.0192	1.0313
9	1.0349	1.0279	1.0147	0.9967	0.9806	0.9696	0.9667	0.9722	0.9854	1.0023	1.0197	1.0316
10	1.0348	1.0276	1.0142	0.9961	0.9802	0.9694	0.9667	0.9726	0.9859	1.0029	1.0202	1.0319
11	1.0347	1.0272	1.0136	0.9956	0.9797	0.9692	0.9668	0.9729	0.9864	1.0035	1.0207	1.0321

12	1.0347	1.0268	1.0131	0.995	0.9793	0.969	0.9668	0.9732	0.9869	1.0041	1.0212	1.0324
13	1.0346	1.0264	1.0125	0.9944	0.9788	0.9687	0.9669	0.9736	0.9875	1.0047	1.0217	1.0326
14	1.0344	1.026	1.0119	0.9938	0.9784	0.9686	0.967	0.9739	0.988	1.0053	1.0222	1.0328
15	1.0343	1.0256	1.0114	0.9932	0.978	0.9684	0.9671	0.9743	0.9885	1.0058	1.0226	1.033
16	1.0342	1.0251	1.0108	0.9927	0.9775	0.9682	0.9672	0.9746	0.9891	1.0064	1.0231	1.0332
17	1.034	1.0247	1.0102	0.9921	0.9771	0.968	0.9673	0.975	0.9896	1.007	1.0235	1.0334
18	1.0339	1.0243	1.0097	0.9915	0.9767	0.9679	0.9674	0.9754	0.9902	1.0076	1.024	1.0336
19	1.0337	1.0238	1.0091	0.991	0.9763	0.9677	0.9675	0.9758	0.9907	1.0082	1.0244	1.0338
20	1.0335	1.0234	1.0085	0.9904	0.9759	0.9676	0.9677	0.9762	0.9913	1.0088	1.0249	1.0339
21	1.0334	1.0229	1.0079	0.9899	0.9755	0.9675	0.9678	0.9766	0.9918	1.0093	1.0253	1.0341
22	1.0332	1.0224	1.0073	0.9893	0.9751	0.9673	0.968	0.977	0.9924	1.0099	1.0257	1.0342
23	1.033	1.022	1.0067	0.9888	0.9748	0.9672	0.9681	0.9774	0.993	1.0105	1.0261	1.0344
24	1.0327	1.0215	1.0062	0.9882	0.9744	0.9671	0.9683	0.9778	0.9935	1.0111	1.0265	1.0345
25	1.0325	1.021	1.0056	0.9877	0.974	0.967	0.9685	0.9782	0.9941	1.0116	1.0269	1.0346

26	1.0323	1.0205	1.005	0.9872	0.9737	0.9669	0.9687	0.9787	0.9947	1.0122	1.0273	1.0347
27	1.032	1.02	1.0044	0.9866	0.9733	0.9669	0.9689	0.9791	0.9953	1.0128	1.0277	1.0348
28	1.0318	1.0195	1.0038	0.9861	0.973	0.9668	0.9691	0.9795	0.9959	1.0133	1.0281	1.0349
29	1.0315		1.0032	0.9856	0.9727	0.9667	0.9693	0.98	0.9964	1.0139	1.0284	1.0349
30	1.0312		1.0026	0.9851	0.9724	0.9667	0.9695	0.9805	0.997	1.0144	1.0288	1.035
31	1.0309		1.002		0.972		0.9698	0.9809		1.015		1.035

Code for calculating eccentricity correction factor using Spencer's expression.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
#Create a numpy array for number of days in a month
days_in_month = np.arange(1,32)
#Create numpy array for number of days in a year
days_in_year = np.arange(1,366)
#Calculate the day angle for each day of the year
day_angle = 2 * np.pi * ((days_in_year - 1)/365)
# Calculate the eccentricity correction factor using spencer's expression
spencer_ecf_calculated = (1.000110 + 0.034221*np.cos(day_angle) + 0.001280 * np.sin(day_angle) + 0.000719*np.cos(2*day_angle) + 0.000077 *  np.sin(2*day_angle))
# Plot the graph of tabulated eccentricity data for each day of the year
plt.figure()
# Plot the tabulated eccentricity data using a red line, and
 #label it as 'tabulated ecf'
plt.plot(days_in_year, spencer_ecf_calculated, "k", label='Spencer ecf')
# Set the x-axis label as "month" and y-axis label as "tabulated ecf"
plt.xlabel("Month")
plt.ylabel("Spencer ecf")
# Show the legend in the lower-right corner
plt.legend(loc='lower right')
# Show the grid lines
```

```
plt.grid()
# Set custom x-axis ticks and labels for months
plt.xticks(np.linspace(0, 365, 13)[:-1],('Jan', 'Feb', 'Mar', 'Apr', 'May','June', 'Jul', 'Aug', 'Sep', 'Oct',
                                         'Nov', 'Dec'))
# Save the plot as an image file named "tabulated_ecf.jpg"
plt.savefig("Spencer_ecf.jpg")
# Show the plot
plt.show()
#We now arrange eccentricity values in tabular form using pandas data frame
#Create an array of values for the number of days in a month
day = np.arange(1,32)
#Partitioning the number of days in a year into months
January = spencer_ecf_calculated[0:31]
February = spencer_ecf_calculated[31:59]
March = spencer_ecf_calculated[59:90]
April = spencer_ecf_calculated[90:120]
May = spencer_ecf_calculated[120:151]
June = spencer_ecf_calculated[151:181]
July = spencer_ecf_calculated[181:212]
August = spencer_ecf_calculated[212:243]
September = spencer_ecf_calculated[243:273]
October = spencer_ecf_calculated[273:304]
November = spencer_ecf_calculated[304:334]
December = spencer_ecf_calculated[334:365]
```

```
# Assigning zeroes to 29, 30 and 31st February, 30th of April, June, September and November to
#fill up the gaps

February=np.append(February,[0.00,0.00,0.00])

April = np.append(April,[0.00])

June = np.append(June,[0.00])

September = np.append(September,[0.00])

November = np.append(November,[0.00])

# Creating a dictionary for months as columns

spencer_declination_dict = {'Jan': list(January),'Feb': list(February),'Mar': list(March), 'Apr':
list(April),'May': list(May),'Jun': list(June),'Jul': list(July), 'Aug': list(August),

'Sep': list(September),'Oct': list(October) ,'Nov': list(November), 'Dec': list(December)}

#Create the data frame using the dictionary

df_spencer_ecf = pd.DataFrame(data=spencer_declination_dict, index=day)

#Save the dataframe to an excel sheet

df_spencer_ecf.to_excel("spencer_ecf.xlsx", index = True)

# Show results

df_spencer_ecf
```

Calculating the Root Mean Square Error from Spencer's expression

```
def root_mean_square_error(np_ecf_tabulated, spencer_ecf_calculated):

    # We find the squared differences of the corresponding values

    squared_differences = np.square(np_ecf_tabulated - spencer_ecf_calculated)

    # We get the mean squared error from the squared differences

    mean_squared_error = np.mean(squared_differences)

    # Calculating the root mean squared error

    root_mean_squared_error = np.sqrt(mean_squared_error)
```

```
    return root_mean_squared_error
Solution = root_mean_square_error(np_ecf_tabulated, spencer_ecf_calculated)
print("Root Mean Square Error:", Solution)
```

Percentage Error from Spencer's expression

```
# Percentage error
def calculate_percentage_error(np_ecf_tabulated, spencer_ecf_calculated):
    # Calculate percentage error for each pair of values
    errors = [(abs(np_ecf_tabulated[i] - spencer_ecf_calculated[i]) /  np_ecf_tabulated[i]) *100
                    for i in range(len(spencer_ecf_calculated))]
    # Calculate the average percentage error
    average_error = sum(errors) / len(errors)
    return average_error
percentage_error = calculate_percentage_error(np_ecf_tabulated,   spencer_ecf_calculated)
print(f"Average Percentage Error: {percentage_error:.2f}%")
```

Eccentricity Correction Factors from Spencer's expression

Date	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	1.0350	1.0306	1.0190	1.0014	0.9845	0.9717	0.9666	0.9700	0.9814	0.9977	1.0156	1.0292
2	1.0351	1.0303	1.0185	1.0008	0.9840	0.9714	0.9666	0.9703	0.9819	0.9983	1.0161	1.0295
3	1.0351	1.0300	1.0179	1.0002	0.9835	0.9711	0.9666	0.9705	0.9824	0.9988	1.0167	1.0299
4	1.0351	1.0297	1.0174	0.9996	0.9830	0.9709	0.9666	0.9708	0.9829	0.9994	1.0172	1.0302
5	1.0351	1.0294	1.0169	0.9990	0.9825	0.9706	0.9666	0.9711	0.9834	1.0000	1.0177	1.0305
6	1.0350	1.0290	1.0164	0.9985	0.9820	0.9703	0.9666	0.9714	0.9839	1.0006	1.0182	1.0308
7	1.0350	1.0287	1.0158	0.9979	0.9816	0.9701	0.9666	0.9717	0.9844	1.0012	1.0188	1.0311
8	1.0350	1.0283	1.0153	0.9973	0.9811	0.9698	0.9666	0.9720	0.9849	1.0018	1.0193	1.0314
9	1.0349	1.0279	1.0147	0.9967	0.9806	0.9696	0.9667	0.9723	0.9854	1.0024	1.0198	1.0316
10	1.0348	1.0276	1.0142	0.9961	0.9802	0.9694	0.9667	0.9726	0.9859	1.0030	1.0203	1.0319
11	1.0347	1.0272	1.0136	0.9955	0.9797	0.9691	0.9668	0.9729	0.9865	1.0036	1.0208	1.0322

12	1.0347	1.0268	1.0131	0.9950	0.9793	0.9689	0.9668	0.9733	0.9870	1.0042	1.0213	1.0324
13	1.0346	1.0264	1.0125	0.9944	0.9788	0.9687	0.9669	0.9736	0.9875	1.0047	1.0217	1.0326
14	1.0344	1.0260	1.0119	0.9938	0.9784	0.9685	0.9670	0.9739	0.9881	1.0053	1.0222	1.0329
15	1.0343	1.0256	1.0114	0.9932	0.9779	0.9684	0.9671	0.9743	0.9886	1.0059	1.0227	1.0331
16	1.0342	1.0251	1.0108	0.9927	0.9775	0.9682	0.9672	0.9747	0.9891	1.0065	1.0232	1.0333
17	1.0340	1.0247	1.0102	0.9921	0.9771	0.9680	0.9673	0.9750	0.9897	1.0071	1.0236	1.0335
18	1.0339	1.0243	1.0096	0.9915	0.9767	0.9679	0.9674	0.9754	0.9902	1.0077	1.0241	1.0336
19	1.0337	1.0238	1.0091	0.9910	0.9763	0.9677	0.9675	0.9758	0.9908	1.0083	1.0245	1.0338
20	1.0335	1.0234	1.0085	0.9904	0.9759	0.9676	0.9677	0.9762	0.9914	1.0088	1.0249	1.0340
21	1.0334	1.0229	1.0079	0.9898	0.9755	0.9674	0.9678	0.9766	0.9919	1.0094	1.0254	1.0341
22	1.0332	1.0224	1.0073	0.9893	0.9751	0.9673	0.9680	0.9770	0.9925	1.0100	1.0258	1.0343
23	1.0330	1.0220	1.0067	0.9888	0.9747	0.9672	0.9682	0.9774	0.9930	1.0106	1.0262	1.0344
24	1.0327	1.0215	1.0061	0.9882	0.9744	0.9671	0.9683	0.9778	0.9936	1.0111	1.0266	1.0345
25	1.0325	1.0210	1.0056	0.9877	0.9740	0.9670	0.9685	0.9783	0.9942	1.0117	1.0270	1.0346

26	1.0323	1.0205	1.0050	0.9871	0.9737	0.9669	0.9687	0.9787	0.9948	1.0123	1.0274	1.0347
27	1.0320	1.0200	1.0044	0.9866	0.9733	0.9669	0.9689	0.9791	0.9953	1.0128	1.0278	1.0348
28	1.0318	1.0195	1.0038	0.9861	0.9730	0.9668	0.9691	0.9796	0.9959	1.0134	1.0281	1.0349
29	1.0315		1.0032	0.9856	0.9727	0.9667	0.9693	0.9800	0.9965	1.0140	1.0285	1.0349
30	1.0312		1.0026	0.9850	0.9723	0.9667	0.9696	0.9805	0.9971	1.0145	1.0289	1.0350
31	1.0309		1.0020		0.9720		0.9698	0.9810		1.0151		1.0350

Duffie and Beckmann Eccentricity Correction Factor

```
#Import numpy alias np
import numpy as np
#Import matplotlib.pyplot alias plt
import matplotlib.pyplot as plt
# Importing the pandas module
import pandas as pd
#Create a numpy array for number of days in a month
days_in_month = np.arange(1,32)
#Create numpy array for number of days in a year
days_in_year = np.arange(1,366)
#Calculate the day angle for each day of the year
day_angle = 2 * np.pi * ((days_in_year - 1)/365)
#Calculating the eccentricity correctiong factor using the Duffie and Beckmann expression
duffie_beckmann_ecf = 1 + 0.033*np.cos(2*np.pi*(days_in_year/365))
# Plot the graph of tabulated eccentricity data for each day of the year
plt.figure()
# Plot the tabulated eccentricity data using a red line, and label it as
# 'tabulated ecf'
plt.plot(days_in_year, duffie_beckmann_ecf, "k", label='Duffie Beckmann ecf')
# Set the x-axis label as "month" and y-axis label as "tabulated ecf"
plt.xlabel("Month")
plt.ylabel("Duffie Beckmann ecf")
# Show the legend in the lower-right corner
```

```
plt.legend(loc='lower right')
# Show the grid lines
plt.grid()
# Set custom x-axis ticks and labels for months
plt.xticks(np.linspace(0, 365, 13)[:-1], ('Jan', 'Feb', 'Mar', 'Apr', 'May','June', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'))
# Save the plot as an image file named "tabulated_ecf.jpg"
plt.savefig("Duffie_Beckmann_ecf.jpg")
# Show the plot
plt.show()
#We now arrange eccentricity values in tabular form using pandas dataframe
#Importing the pandas module
import pandas as pd
#Create an array of values for the number of days in a month
day = np.arange(1,32)
#Partitioning the number of days in a year into months
January = duffie_beckmann_ecf[0:31]
February = duffie_beckmann_ecf[31:59]
March = duffie_beckmann_ecf[59:90]
April = duffie_beckmann_ecf[90:120]
May = duffie_beckmann_ecf[120:151]
June = duffie_beckmann_ecf[151:181]
July = duffie_beckmann_ecf[181:212]
August = duffie_beckmann_ecf[212:243]
```

```
September = duffie_beckmann_ecf[243:273]

October = duffie_beckmann_ecf[273:304]

November = duffie_beckmann_ecf[304:334]

December = duffie_beckmann_ecf[334:365]

# Assigning zeroes to 29, 30 and 31st February, 30th of April, June, September and November

February=np.append(February,[0.00,0.00,0.00])

April = np.append(April,[0.00])

June = np.append(June,[0.00])

September = np.append(September,[0.00])

November = np.append(November,[0.00])

# Creating a dictionary for months as columns

duffie_beckmann_declination_data = {'Jan': January, 'Feb': February,'Mar': March, 'Apr': April, 'May': May, 'Jun': June, 'Jul': July, 'Aug': August, 'Sep': September, 'Oct': October, 'Nov': November, 'Dec': December}

#Create the data frame using the dictionary

df_duffie_beckmann_ecf = pd.DataFrame(data=duffie_beckmann_declination_data, index=day)

#round the values to 4 decimal places

df_duffie_beckmann_ecf_rounded = df_duffie_beckmann_ecf.round(4)

#Save te dataframe to an excel sheet

df_duffie_beckmann_ecf_rounded.to_excel('duffie_beckmann_ecf_output.xlsx')

# Show results

df_duffie_beckmann_ecf_rounded
```

Root Mean Square Error from Duffie and Beckmann expression

```
def root_mean_square_error(np_ecf_tabulated, duffie_beckmann_ecf):

    # We find the squared differences of the corresponding values
```

```
    squared_differences = np.square(np_ecf_tabulated - duffie_beckmann_ecf)
    # We get the mean squared error from the squared differences
    mean_squared_error = np.mean(squared_differences)
    # Calculating the root mean squared error
    root_mean_squared_error = np.sqrt(mean_squared_error)
    return root_mean_squared_error
Solution = root_mean_square_error(np_ecf_tabulated, duffie_beckmann_ecf)
print("Root Mean Square Error:", Solution)
```

Percentage Error

```
# Percentage error
def calculate_percentage_error(np_ecf_tabulated, duffie_beckmann_ecf):
    # Calculate percentage error for each pair of values
    errors = [(abs(np_ecf_tabulated[i] - duffie_beckmann_ecf[i]) /  np_ecf_tabulated[i])  *
                100 for i in range(len(duffie_beckmann_ecf))]
    # Calculate the average percentage error
    average_error = sum(errors) / len(errors)
    return average_error
percentage_error = calculate_percentage_error(np_ecf_tabulated,   duffie_beckmann_ecf)
print(f"Average Percentage Error: {percentage_error:.2f}%"
```

Graphical comparison

```
plt.figure()
plt.plot(days_in_year,np_ecf_tabulated, "r", label='tabulated ecf')
plt.plot(days_in_year,spencer_ecf_calculated, "k-.", label='spencer ecf')
plt.plot(days_in_year,duffie_beckmann_ecf, "b-.", label='duffie beckman ecf')
```

```
plt.xlabel("Month")
plt.ylabel("eccentric correction factor")
plt.legend(loc='lower right')
plt.grid()
plt.xticks(np.linspace(0,365,13)[:-1],('Jan','Feb','Mar','Apr','May','June','Jul','Aug','Sep','Oct',
                                       'Nov', 'Dec'))
plt.savefig("tabulated_spencer_duffie_beckman_ecf.jpg")
plt.show()
```

Eccentricity Correction Factors from Duffie and Beckmann's expression

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	1.0330	1.0281	1.0169	1.0001	0.9838	0.9714	0.9670	0.9714	0.9838	1.0001	1.0169	1.0287
2	1.0330	1.0278	1.0164	0.9996	0.9833	0.9712	0.9670	0.9717	0.9843	1.0007	1.0174	1.0290
3	1.0330	1.0275	1.0159	0.9990	0.9828	0.9709	0.9670	0.9720	0.9848	1.0013	1.0179	1.0292
4	1.0329	1.0272	1.0154	0.9984	0.9824	0.9706	0.9670	0.9723	0.9853	1.0018	1.0183	1.0295
5	1.0329	1.0269	1.0149	0.9979	0.9819	0.9704	0.9671	0.9727	0.9858	1.0024	1.0188	1.0297
6	1.0328	1.0265	1.0144	0.9973	0.9814	0.9701	0.9671	0.9730	0.9864	1.0030	1.0193	1.0300
7	1.0328	1.0262	1.0139	0.9967	0.9809	0.9699	0.9671	0.9733	0.9869	1.0035	1.0197	1.0302
8	1.0327	1.0258	1.0134	0.9962	0.9805	0.9697	0.9672	0.9736	0.9874	1.0041	1.0202	1.0304
9	1.0326	1.0255	1.0129	0.9956	0.9800	0.9694	0.9673	0.9740	0.9879	1.0047	1.0206	1.0307
10	1.0325	1.0251	1.0123	0.9950	0.9796	0.9692	0.9674	0.9743	0.9885	1.0052	1.0211	1.0309
11	1.0324	1.0247	1.0118	0.9945	0.9791	0.9690	0.9674	0.9747	0.9890	1.0058	1.0215	1.0311
12	1.0323	1.0244	1.0113	0.9939	0.9787	0.9688	0.9675	0.9751	0.9895	1.0064	1.0219	1.0313

13	1.0322	1.0240	1.0107	0.9934	0.9783	0.9687	0.9676	0.9754	0.9901	1.0069	1.0224	1.0314
14	1.0320	1.0236	1.0102	0.9928	0.9778	0.9685	0.9678	0.9758	0.9906	1.0075	1.0228	1.0316
15	1.0319	1.0232	1.0097	0.9923	0.9774	0.9683	0.9679	0.9762	0.9912	1.0080	1.0232	1.0318
16	1.0318	1.0228	1.0091	0.9917	0.9770	0.9682	0.9680	0.9766	0.9917	1.0086	1.0236	1.0319
17	1.0316	1.0224	1.0086	0.9912	0.9766	0.9680	0.9682	0.9770	0.9923	1.0091	1.0240	1.0320
18	1.0314	1.0219	1.0080	0.9906	0.9762	0.9679	0.9683	0.9774	0.9928	1.0097	1.0244	1.0322
19	1.0313	1.0215	1.0075	0.9901	0.9758	0.9678	0.9685	0.9778	0.9934	1.0102	1.0247	1.0323
20	1.0311	1.0211	1.0069	0.9895	0.9754	0.9676	0.9687	0.9783	0.9939	1.0107	1.0251	1.0324
21	1.0309	1.0206	1.0064	0.9890	0.9751	0.9675	0.9688	0.9787	0.9945	1.0113	1.0255	1.0325
22	1.0307	1.0202	1.0058	0.9885	0.9747	0.9674	0.9690	0.9791	0.9950	1.0118	1.0258	1.0326
23	1.0304	1.0197	1.0052	0.9879	0.9743	0.9674	0.9692	0.9796	0.9956	1.0123	1.0262	1.0327
24	1.0302	1.0193	1.0047	0.9874	0.9740	0.9673	0.9694	0.9800	0.9962	1.0129	1.0265	1.0328
25	1.0300	1.0188	1.0041	0.9869	0.9736	0.9672	0.9697	0.9805	0.9967	1.0134	1.0269	1.0328
26	1.0297	1.0183	1.0035	0.9864	0.9733	0.9671	0.9699	0.9809	0.9973	1.0139	1.0272	1.0329

27	1.0295	1.0179	1.0030	0.9858	0.9730	0.9671	0.9701	0.9814	0.9979	1.0144	1.0275	1.0329
28	1.0292	1.0174	1.0024	0.9853	0.9727	0.9671	0.9704	0.9819	0.9984	1.0149	1.0278	1.0330
29	1.0290		1.0018	0.9848	0.9723	0.9670	0.9706	0.9824	0.9990	1.0154	1.0281	1.0330
30	1.0287		1.0013	0.9843	0.9720	0.9670	0.9709	0.9828	0.9996	1.0159	1.0284	1.0330
31	1.0284		1.0007		0.9717		0.9712	0.9833		1.0164		1.0330

Appendix 4: Tables and Computer Codes for Declination Angles

Declination Angles obtained from Iqbal (1983)

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	-23.07	-17.28	-7.78	4.36	14.93	22.02	23.20	18.20	8.51	-2.95	-14.26	-21.74
2	-22.99	-17.00	-7.40	4.75	15.24	22.15	23.13	17.94	8.14	-3.33	-14.58	-21.90
3	-22.90	-16.71	-7.02	5.13	15.54	22.29	23.06	17.69	7.78	-3.72	-14.9	-22.05
4	-22.80	-16.41	-6.63	5.51	15.83	22.41	22.98	17.42	7.41	-4.11	-15.22	-22.19
5	-22.70	-16.11	-6.25	5.89	16.12	22.53	22.89	17.16	7.04	-4.50	-15.53	-22.32
6	-22.59	-15.81	-5.86	6.27	16.41	22.64	22.80	16.89	6.67	-4.88	-15.83	-22.45
7	-22.47	-15.50	-5.47	6.65	16.69	22.74	22.70	16.61	6.30	-5.27	-16.13	-22.57
8	-22.34	-15.18	-5.08	7.03	16.96	22.84	22.59	16.33	5.93	-5.65	-16.43	-22.68
9	-22.21	-14.87	-4.69	7.40	17.24	22.93	22.48	16.05	5.55	-6.03	-16.72	-22.79
10	-22.07	-14.54	-4.30	7.77	17.50	23.01	22.36	15.76	5.17	-6.41	-17.01	-22.28
11	-21.92	-14.22	-3.90	8.14	17.77	23.09	22.23	15.46	4.80	-6.79	-17.29	-22.98

12	-21.76	-13.89	-3.51	8.51	18.02	23.16	22.10	15.17	4.42	-7.17	-17.57	-23.06
13	-21.60	-13.55	-3.12	8.87	18.28	23.23	21.96	14.87	4.03	-7.55	-17.84	-23.13
14	-21.43	-13.22	-2.72	9.24	18.52	23.28	21.81	14.56	3.65	-7.92	-18.11	-23.20
15	-21.25	-12.87	-2.33	9.60	18.77	23.33	21.66	14.25	3.27	-8.30	-18.37	-23.26
16	-21.07	-12.53	-1.93	9.95	19.00	23.38	21.50	13.94	2.88	-8.67	-18.62	-23.31
17	-20.88	-12.18	-1.54	10.31	19.23	23.41	21.34	13.62	2.50	-9.04	-18.87	-23.36
18	-20.68	-11.83	-1.14	10.66	19.46	23.44	21.17	13.30	2.11	-9.40	-19.12	-23.39
19	-20.48	-11.47	-0.74	11.01	19.68	23.47	20.99	12.98	1.72	-9.77	-19.36	-23.42
20	-20.27	-11.12	-0.35	11.35	19.90	23.48	20.81	12.66	1.34	-10.13	-19.59	-23.44
21	-20.05	-10.76	0.05	11.70	20.10	23.49	20.63	12.33	0.95	-10.49	-19.82	-23.46
22	-19.83	-10.39	0.44	12.04	20.31	23.49	20.43	11.99	0.56	-10.85	-20.04	-23.46
23	-19.60	-10.03	0.84	12.37	20.51	23.49	20.23	11.66	0.17	-11.21	-20.25	-23.46
24	-19.37	-9.66	1.23	12.71	20.70	23.47	20.03	11.32	-0.22	-11.56	-20.46	-23.45
25	-19.13	-9.29	1.63	13.04	20.88	23.46	19.82	10.98	-0.61	-11.91	-20.67	-23.43

26	-18.88	-8.91	2.02	13.36	21.07	23.43	19.60	10.63	-1.00	-12.25	-20.86	-23.4
27	-18.63	-8.54	2.41	13.68	21.24	23.40	19.38	10.28	-1.39	-12.60	-21.05	-23.37
28	-18.37	-8.16	2.80	14.00	21.41	23.36	19.15	9.93	-1.78	-12.94	-21.23	-23.33
29	-18.11		3.19	14.32	21.57	23.31	18.92	9.58	-2.17	-13.27	-21.41	-23.28
30	-17.84		3.58	14.63	21.73	23.26	18.68	9.22	-2.56	-13.61	-21.58	-23.22
31	-17.56		3.97		21.87		18.44	8.87		-13.94		-23.16

Calculating Declination angles using Spencer’s expression

```
#Import numpy alias np
import numpy as np
#Import matplotlib.pyplot alias plt
import matplotlib.pyplot as plt
# Importing the pandas module
import pandas as pd
# Import pi from numpy
from numpy import pi
#Create a numpy array for number of days in a month
days_in_month = np.arange(1,32)
#Create numpy array for number of days in a year
days_in_year = np.arange(1,366)
#Calculate the day angle for each day of the year
day_angle = 2 * np.pi * ((days_in_year - 1)/365)
#We now use the spencer's expression to calculate the declination angle
spencer_declination = ((0.006918-0.399912*np.cos(day_angle) +0.070257 *np.sin(day_angle)
                                -0.006758*np.cos(2*day_angle)+0.000907 *  np.sin(2*day_angle)
                        -0.002697*np.cos(3*day_angle)+0.00148* np.sin(3*day_angle)))*(180/np.pi)
#Plot the declination angle for each day of the year
plt.figure()
# Plot the declination angle using a red line, and label it as 'spencer declination angle'
plt.plot(days_in_year,spencer_declination, "k", label='spencer declination angle')
# Set the x-axis label as "Month" and y-axis label as "Declination Angle"
```

```
plt.xlabel("Month")
plt.ylabel("Declination Angle")
# Set the legend
plt.legend(loc='lower right')
# Show the grid lines
plt.grid()
# Set custom x-axis ticks and labels for months
plt.xticks(np.linspace(0,365,13)[:-1],('Jan','Feb','Mar','Apr','May','June','Jul','Aug','Sep','Oct',
                                       'Nov','Dec') )
# Save the plot as an image file named "tabulated_ecf.jpg"
plt.savefig("spencer_declination.jpg")
# Display the plot
plt.show()
#We now arrange eccentricity values in tabular form using pandas dataframe
#Partitioning the number of days in a year into months
January = spencer_declination[0:31]
February = spencer_declination[31:59]
March = spencer_declination[59:90]
April = spencer_declination[90:120]
May = spencer_declination[120:151]
June = spencer_declination[151:181]
July = spencer_declination[181:212]
August = spencer_declination[212:243]
September = spencer_declination[243:273]
```

```
October = spencer_declination[273:304]

November = spencer_declination[304:334]

December = spencer_declination[334:365]

# Appending zeroes to non 31 day months

February=np.append(February,[0.00,0.00,0.00])

April = np.append(April,[0.00])

June = np.append(June,[0.00])

September = np.append(September,[0.00])

November = np.append(November,[0.00])

#Creating a pandas data frame

# Creating a dictionary for months as columns

spencer_declination_dict = {'Jan': list(January), 'Feb': list(February),'Mar': list(March), 'Apr': list(April),'May': list(May),'Jun': list(June),'Jul': list(July), 'Aug': list(August),

'Sep': list(September),'Oct': list(October),'Nov': list(November), 'Dec': list(December)}

#Create the data frame using the dictionary

df_spencer_declination = pd.DataFrame(data=spencer_declination_dict, index=days_in_month)

#Save te dataframe to an excel sheet

df_spencer_declination.to_excel("spencer_declination_angle.xlsx",index =True)

# Show results

df_spencer_declination
```

Declination angles from Spencer's expression

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	-23.06	-17.34	-7.88	4.24	14.83	21.95	23.18	18.22	8.57	-2.87	-14.19	-21.69
2	-22.98	-17.05	-7.50	4.63	15.13	22.09	23.11	17.97	8.21	-3.25	-14.51	-21.85
3	-22.89	-16.76	-7.12	5.01	15.43	22.22	23.04	17.72	7.85	-3.64	-14.83	-22.00
4	-22.80	-16.47	-6.73	5.40	15.73	22.34	22.96	17.46	7.48	-4.03	-15.14	-22.14
5	-22.70	-16.17	-6.35	5.78	16.02	22.46	22.88	17.19	7.11	-4.42	-15.46	-22.27
6	-22.59	-15.87	-5.96	6.16	16.31	22.58	22.78	16.92	6.74	-4.80	-15.76	-22.40
7	-22.47	-15.56	-5.57	6.53	16.59	22.68	22.69	16.65	6.37	-5.19	-16.06	-22.52
8	-22.35	-15.25	-5.19	6.91	16.87	22.78	22.58	16.37	6.00	-5.57	-16.36	-22.64
9	-22.21	-14.93	-4.79	7.28	17.14	22.87	22.47	16.09	5.62	-5.95	-16.65	-22.74
10	-22.07	-14.61	-4.40	7.66	17.41	22.96	22.35	15.80	5.25	-6.33	-16.94	-22.84
11	-21.93	-14.29	-4.01	8.03	17.67	23.04	22.23	15.51	4.87	-6.71	-17.22	-22.93
12	-21.77	-13.96	-3.62	8.39	17.93	23.11	22.09	15.21	4.49	-7.09	-17.50	-23.01

13	-21.61	-13.63	-3.23	8.76	18.18	23.18	21.96	14.91	4.11	-7.47	-17.77	-23.09
14	-21.45	-13.29	-2.83	9.12	18.43	23.23	21.81	14.61	3.73	-7.84	-18.04	-23.16
15	-21.27	-12.95	-2.44	9.48	18.67	23.29	21.66	14.30	3.34	-8.22	-18.30	-23.22
16	-21.09	-12.61	-2.04	9.84	18.91	23.33	21.51	13.99	2.96	-8.59	-18.56	-23.27
17	-20.90	-12.26	-1.65	10.19	19.14	23.37	21.35	13.67	2.57	-8.96	-18.81	-23.32
18	-20.71	-11.91	-1.25	10.55	19.37	23.40	21.18	13.36	2.19	-9.33	-19.06	-23.35
19	-20.51	-11.56	-0.86	10.89	19.59	23.42	21.00	13.03	1.80	-9.69	-19.30	-23.38
20	-20.30	-11.20	-0.46	11.24	19.81	23.44	20.82	12.71	1.41	-10.06	-19.53	-23.41
21	-20.09	-10.84	-0.07	11.58	20.02	23.45	20.64	12.38	1.03	-10.42	-19.76	-23.42
22	-19.87	-10.48	0.33	11.92	20.23	23.46	20.44	12.05	0.64	-10.77	-19.98	-23.43
23	-19.64	-10.12	0.72	12.26	20.43	23.45	20.25	11.71	0.25	-11.13	-20.20	-23.42
24	-19.41	-9.75	1.12	12.60	20.62	23.44	20.04	11.38	-0.14	-11.48	-20.41	-23.41
25	-19.17	-9.38	1.51	12.93	20.81	23.42	19.84	11.03	-0.53	-11.83	-20.61	-23.40
26	-18.92	-9.01	1.90	13.25	20.99	23.40	19.62	10.69	-0.92	-12.18	-20.81	-23.37

27	-18.67	-8.63	2.30	13.57	21.16	23.37	19.40	10.34	-1.31	-12.52	-21.0	-23.34
28	-18.42	-8.26	2.69	13.89	21.33	23.33	19.18	9.99	-1.70	-12.86	-21.18	-23.30
29	-18.15		3.08	14.21	21.50	23.29	18.95	9.64	-2.09	-13.20	-21.36	-23.25
30	-17.89		3.47	14.52	21.65	23.24	18.71	9.29	-2.48	-13.53	-21.53	-23.19
31	-17.61		3.86		21.80		18.47	8.93		-13.86		-23.13

Calculating the Mean Square Root Error for Spencer's expression

```
def root_mean_square_error(tabulated_declination, spencer_declination):
    # We find the squared differences of the corresponding values
    squared_differences = np.square(tabulated_declination - \spencer_declination)
    # We get the mean squared error from the squared differences
    mean_squared_error = np.mean(squared_differences)
    # Calculating the root mean squared error
    root_mean_squared_error = np.sqrt(mean_squared_error)
    return root_mean_squared_error
Solution = root_mean_square_error(tabulated_declination, spencer_declination)
print("Root Mean Square Error:", Solution)
```

Spencer's expression Percentage Error

```
# Percentage error
def calculate_percentage_error(tabulated_declination, spencer_declination):
    # Calculate percentage error for each pair of values
    errors = [(abs(tabulated_declination[i] - spencer_declination[i]) / tabulated_declination[i]) * 100
                for i in range(len(spencer_declination))]
    # Calculate the average percentage error
    average_error = sum(errors) / len(errors)
    return average_error
percentage_error = calculate_percentage_error(tabulated_declination, spencer_declination)
print(f"Average Percentage Error: {percentage_error:.2f}%")
```

Declination using Perrin de Brichambaut’s expression

```
#Import numpy alias np
import numpy as np
#Import matplotlib.pyplot alias plt
import matplotlib.pyplot as plt
# Importing the pandas module
import pandas as pd
# Importing pi from numpy
from numpy import pi
#Create a numpy array for number of days in a month
days_in_month = np.arange(1,32)
#Create numpy array for number of days in a year
days_in_year = np.arange(1,366)
#Calculate the day angle for each day of the year
day_angle = 2 * np.pi * ((days_in_year - 1)/365)
# Calculate the eccentricity correction factor using Perrin de Brichambaut expression
perrin_de_brichambaut =np.arcsin(0.4*np.sin((360/365)*(days_in_year -82)*pi/180))*180/pi
#Plot the declination angle for each day of the year
plt.figure()
# Plot the declination angle and label it as 'Perrin de Brichambaut'
plt.plot(days_in_year,perrin_de_brichambaut, label='Perrin de Brichambaut')
# Set the x-axis label as "Month" and y-axis label as "Declination Angle"
plt.xlabel("Month")
plt.ylabel("Declination Angle")
```

```
# Set the x-axis label as "month" and y-axis label as "tabulated ecf"
plt.legend(loc='lower right')
# Show the grid lines
plt.grid()
# Set custom x-axis ticks and labels for months
plt.xticks(np.linspace(0,365,13)[:-1],('Jan','Feb','Mar','Apr','May','June','Jul','Aug','Sep','Oct',
                                       'Nov','Dec') )
# Save the plot as an image file named "tabulated_ecf.jpg"
plt.savefig("perrin_de_brichambaut.jpg")
# Display the plot
plt.show()
#We now arrange eccentricity values in tabular form using pandas dataframe
#Partitioning the number of days in a year into months
January = perrin_de_brichambaut[0:31]
February = perrin_de_brichambaut[31:59]
March = perrin_de_brichambaut[59:90]
April = perrin_de_brichambaut[90:120]
May = perrin_de_brichambaut[120:151]
June = perrin_de_brichambaut[151:181]
July = perrin_de_brichambaut[181:212]
August = perrin_de_brichambaut[212:243]
September = perrin_de_brichambaut[243:273]
October = perrin_de_brichambaut[273:304]
November = perrin_de_brichambaut[304:334]
```

```
December = perrin_de_brichambaut[334:365]

# Assigning zeroes to 29, 30 and 31st February, 30th of April, June,September and November to fill up the gaps

February=np.append(February,[0.00,0.00,0.00])

April = np.append(April,[0.00])

June = np.append(June,[0.00])

September = np.append(September,[0.00])

November = np.append(November,[0.00])

# Creating a dictionary for months as columns

perrin_de_brichambaut_dict = {'Jan': list(January), 'Feb': list(February),'Mar': list(March), 'Apr': list(April),'May': list(May),'Jun': list(June),'Jul': list(July), 'Aug': list(August),'Sep': list(September),'Oct': list(October),'Nov': list(November), 'Dec': list(December)}

#Create the data frame using the dictionary

df_perrin_de_brichambaut = pd.DataFrame(data=perrin_de_brichambaut_dict, index = days_in_month)

#Save the dataframe to an excel sheet

df_perrin_de_brichambaut.to_excel("perrin_de_brichambaut.xlsx", index = True)

#Display the table

df_perrin_de_brichambaut
```

Declination angles from Perrin de Brichambaut's expression

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	-23.19	-17.66	-8.50	3.54	14.41	21.93	23.30	18.06	7.95	-3.73	-14.88	-22.01
2	-23.11	-17.38	-8.13	3.93	14.72	22.08	23.23	17.79	7.57	-4.12	-15.19	-22.15
3	-23.03	-17.11	-7.76	4.32	15.04	22.22	23.15	17.52	7.19	-4.51	-15.50	-22.29
4	-22.93	-16.83	-7.38	4.71	15.35	22.36	23.07	17.25	6.82	-4.90	-15.80	-22.42
5	-22.83	-16.54	-7.01	5.09	15.65	22.49	22.98	16.97	6.44	-5.29	-16.10	-22.55
6	-22.72	-16.25	-6.63	5.48	15.95	22.61	22.88	16.68	6.05	-5.67	-16.39	-22.67
7	-22.61	-15.95	-6.25	5.86	16.25	22.72	22.78	16.39	5.67	-6.05	-16.68	-22.78
8	-22.49	-15.65	-5.86	6.25	16.54	22.83	22.67	16.10	5.29	-6.44	-16.97	-22.88
9	-22.36	-15.35	-5.48	6.63	16.83	22.93	22.55	15.80	4.90	-6.82	-17.25	-22.98
10	-22.22	-15.04	-5.09	7.01	17.11	23.03	22.42	15.50	4.51	-7.19	-17.52	-23.07
11	-22.08	-14.72	-4.71	7.38	17.38	23.11	22.29	15.19	4.12	-7.57	-17.79	-23.15
12	-21.93	-14.41	-4.32	7.76	17.66	23.19	22.15	14.88	3.73	-7.95	-18.06	-23.23

13	-21.78	-14.09	-3.93	8.13	17.92	23.26	22.01	14.57	3.34	-8.32	-18.32	-23.30
14	-21.62	-13.76	-3.54	8.50	18.19	23.33	21.86	14.25	2.95	-8.69	-18.57	-23.36
15	-21.45	-13.43	-3.15	8.87	18.44	23.38	21.70	13.92	2.56	-9.06	-18.82	-23.41
16	-21.27	-13.10	-2.76	9.24	18.70	23.43	21.53	13.60	2.17	-9.42	-19.06	-23.46
17	-21.09	-12.77	-2.36	9.61	18.94	23.48	21.36	13.27	1.77	-9.79	-19.30	-23.49
18	-20.90	-12.43	-1.97	9.97	19.18	23.51	21.18	12.93	1.38	-10.15	-19.53	-23.53
19	-20.71	-12.09	-1.58	10.33	19.42	23.54	21.00	12.60	0.99	-10.51	-19.76	-23.55
20	-20.51	-11.74	-1.18	10.69	19.65	23.56	20.81	12.26	0.59	-10.86	-19.98	-23.57
21	-20.30	-11.39	-0.79	11.04	19.87	23.57	20.61	11.91	0.20	-11.22	-20.2	-23.58
22	-20.09	-11.04	-0.39	11.39	20.09	23.58	20.41	11.57	-0.20	-11.57	-20.41	-23.58
23	-19.87	-10.69	0.00	11.74	20.30	23.58	20.20	11.22	-0.59	-11.91	-20.61	-23.57
24	-19.65	-10.33	0.39	12.09	20.51	23.57	19.98	10.86	-0.99	-12.26	-20.81	-23.56
25	-19.42	-9.97	0.79	12.43	20.71	23.55	19.76	10.51	-1.38	-12.60	-21.00	-23.54
26	-19.18	-9.61	1.18	12.77	20.90	23.53	19.53	10.15	-1.77	-12.93	-21.18	-23.51

27	-18.94	-9.24	1.58	13.10	21.09	23.49	19.30	9.79	-2.17	-13.27	-21.36	-23.48
28	-18.70	-8.87	1.97	13.43	21.27	23.46	19.06	9.42	-2.56	-13.60	-21.53	-23.43
29	-18.44		2.36	13.76	21.45	23.41	18.82	9.06	-2.95	-13.92	-21.70	-23.38
30	-18.19		2.76	14.09	21.62	23.36	18.57	8.69	-3.34	-14.25	-21.86	-23.33
31	-17.92		3.15		21.78		18.32	8.32		-14.57		-23.26

Mean Square Root Error for Perrin de Brichambaut's expression

```
def root_mean_square_error(tabulated_declination, perrin_de_brichambaut):
    # We find the squared differences of the corresponding values
    squared_differences = np.square(tabulated_declination -  perrin_de_brichambaut)
    # We get the mean squared error from the squared differences
    mean_squared_error = np.mean(squared_differences)
    # Calculating the root mean squared error
    root_mean_squared_error = np.sqrt(mean_squared_error)
    return root_mean_squared_error
Solution = root_mean_square_error(tabulated_declination, perrin_de_brichambaut)
print("Root Mean Square Error:", Solution)
```

Perrin de Brichambaut's expression Percentage Error

```
# Percentage error
def calculate_percentage_error(tabulated_declination, perrin_de_brichambaut):
    # Calculate percentage error for each pair of values
    errors = [(abs(tabulated_declination[i] - perrin_de_brichambaut[i]) / tabulated_declination[i]) *
                    100 for i in range(len(perrin_de_brichambaut))]
    # Calculate the average percentage error
    average_error = sum(errors) / len(errors)
    return average_error
percentage_error = calculate_percentage_error(tabulated_declination, perrin_de_brichambaut)
print(f"Average Percentage Error: {percentage_error:.2f}%")
```

Calculating the Declination angles using Coopers expression

```
#Import numpy alias np
import numpy as np
#Import matplotlib.pyplot alias plt
import matplotlib.pyplot as plt
# Importing the pandas module
import pandas as pd
# Importing pi from numpy
from numpy import pi
#Create a numpy array for number of days in a month
days_in_month = np.arange(1,32)
#Create numpy array for number of days in a year
days_in_year = np.arange(1,366)
#Calculate the day angle for each day of the year
day_angle = 2 * np.pi * ((days_in_year - 1)/365)
# Calculate the eccentricity correction factor using Cooper's expression
cooper_declination = []
for day in days_in_year:
    declination = 23.45*np.sin(((360/365)*(day + 284))*pi/180)
    cooper_declination.append(declination)
#Plot the declination angle for each day of the year
plt.figure()
# Plot the declination angle and label it as 'Cooper Declination'
plt.plot(days_in_year,cooper_declination, label='Cooper Declination Angle')
```

```
# Set the x-axis label as "Month" and y-axis label as "Declination Angle"
plt.xlabel("Month")
plt.ylabel("Declination Angle")
# Set the x-axis label as "month" and y-axis label as "tabulated ecf"
plt.legend(loc='lower right')
# Show the grid lines
plt.grid()
# Set custom x-axis ticks and labels for months
plt.xticks(np.linspace(0,365,13)[:-1],('Jan','Feb','Mar','Apr','May','June',Jul','Aug','Sep','Oct',
                                        'Nov', 'Dec') )
# Save the plot as an image file named "tabulated_ecf.jpg"
plt.savefig("cooper_declination.jpg")
# Display the plot
plt.show()
#We now arrange eccentricity values in tabular form using pandas dataframe
#Partitioning the number of days in a year into months
January = cooper_declination[0:31]
February = cooper_declination[31:59]
March = cooper_declination[59:90]
April = cooper_declination[90:120]
May = cooper_declination[120:151]
June = cooper_declination[151:181]
July = cooper_declination[181:212]
August = cooper_declination[212:243]
```

```
September = cooper_declination[243:273]

October = cooper_declination[273:304]

November = cooper_declination[304:334]

December = cooper_declination[334:365]

# Assigning zeroes to 29, 30 and 31st February, 30th of April, June, September

# and November to fill up the gaps

February=np.append(February,[0.00,0.00,0.00])

April = np.append(April,[0.00])

June = np.append(June,[0.00])

September = np.append(September,[0.00])

November = np.append(November,[0.00])

#Creating a dictionary for months as columns

# Creating a dictionary for months as columns

cooper_declination_dict = {'Jan': list(January), 'Feb': list(February),'Mar': list(March), 'Apr':
list(April),'May': list(May),'Jun': list(June),'Jul': list(July), 'Aug': list(August),

'Sep': list(September),'Oct': list(October),'Nov': list(November), 'Dec': list(December)}

#Create the data frame using the dictionary

df_cooper_declination = pd.DataFrame(data=cooper_declination_dict, index=days_in_month)

#Save the dataframe to an excel sheet

df_cooper_declination.to_excel("cooper_declination.xlsx", index = True)

#Display the results

df_cooper_declination
```

Declination angles from Cooper's expression

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	-23.01	-17.52	-8.29	4.02	14.90	22.04	23.12	17.91	7.72	-4.22	-15.36	-22.11
2	-22.93	-17.25	-7.91	4.41	15.21	22.17	23.05	17.65	7.34	-4.61	-15.67	-22.24
3	-22.84	-16.97	-7.53	4.81	15.52	22.30	22.97	17.38	6.96	-5.01	-15.96	-22.36
4	-22.75	-16.69	-7.15	5.20	15.82	22.42	22.89	17.11	6.57	-5.40	-16.26	-22.48
5	-22.65	-16.40	-6.76	5.60	16.11	22.54	22.80	16.83	6.18	-5.79	-16.55	-22.59
6	-22.54	-16.11	-6.38	5.99	16.40	22.65	22.70	16.55	5.79	-6.18	-16.83	-22.70
7	-22.42	-15.82	-5.99	6.38	16.69	22.75	22.59	16.26	5.40	-6.57	-17.11	-22.80
8	-22.30	-15.52	-5.60	6.76	16.97	22.84	22.48	15.96	5.01	-6.96	-17.38	-22.89
9	-22.17	-15.21	-5.20	7.15	17.25	22.93	22.36	15.67	4.61	-7.34	-17.65	-22.97
10	-22.04	-14.90	-4.81	7.53	17.52	23.01	22.24	15.36	4.22	-7.72	-17.91	-23.05
11	-21.90	-14.59	-4.41	7.91	17.78	23.09	22.11	15.06	3.82	-8.10	-18.17	-23.12
12	-21.75	-14.27	-4.02	8.29	18.04	23.15	21.97	14.74	3.42	-8.48	-18.42	-23.18

13	-21.60	-13.95	-3.62	8.67	18.30	23.21	21.83	14.43	3.02	-8.86	-18.67	-23.24
14	-21.44	-13.62	-3.22	9.04	18.55	23.27	21.67	14.11	2.62	-9.23	-18.91	-23.29
15	-21.27	-13.29	-2.82	9.41	18.79	23.31	21.52	13.78	2.22	-9.60	-19.15	-23.34
16	-21.10	-12.95	-2.42	9.78	19.03	23.35	21.35	13.45	1.81	-9.97	-19.38	-23.37
17	-20.92	-12.62	-2.02	10.15	19.26	23.39	21.18	13.12	1.41	-10.33	-19.6	-23.40
18	-20.73	-12.27	-1.61	10.51	19.49	23.41	21.01	12.79	1.01	-10.69	-19.82	-23.42
19	-20.54	-11.93	-1.21	10.87	19.71	23.43	20.82	12.45	0.61	-11.05	-20.03	-23.44
20	-20.34	-11.58	-0.81	11.23	19.93	23.44	20.64	12.10	0.20	-11.40	-20.24	-23.45
21	-20.14	-11.23	-0.40	11.58	20.14	23.45	20.44	11.75	-0.20	-11.75	-20.44	-23.45
22	-19.93	-10.87	0.00	11.93	20.34	23.45	20.24	11.40	-0.61	-12.10	-20.64	-23.44
23	-19.71	-10.51	0.40	12.27	20.54	23.44	20.03	11.05	-1.01	-12.45	-20.82	-23.43
24	-19.49	-10.15	0.81	12.62	20.73	23.42	19.82	10.69	-1.41	-12.79	-21.01	-23.41
25	-19.26	-9.78	1.21	12.95	20.92	23.40	19.60	10.33	-1.81	-13.12	-21.18	-23.39
26	-19.03	-9.41	1.61	13.29	21.10	23.37	19.38	9.97	-2.22	-13.45	-21.35	-23.35

27	-18.79	-9.04	2.02	13.62	21.27	23.34	19.15	9.60	-2.62	-13.78	-21.52	-23.31
28	-18.55	-8.67	2.42	13.95	21.44	23.29	18.91	9.23	-3.02	-14.11	-21.67	-23.27
29	-18.30		2.82	14.27	21.60	23.24	18.67	8.86	-3.42	-14.43	-21.83	-23.21
30	-18.04		3.22	14.59	21.75	23.18	18.42	8.48	-3.82	-14.74	-21.97	-23.15
31	-17.78		3.62		21.90		18.17	8.10		-15.06		-23.09

Mean Square Root Error for Cooper's expression

```
def root_mean_square_error(tabulated_declination, cooper_declination):
    # We find the squared differences of the corresponding values
    squared_differences = np.square(tabulated_declination -  cooper_declination)
    # We get the mean squared error from the squared differences
    mean_squared_error = np.mean(squared_differences)
    # Calculating the root mean squared error
    root_mean_squared_error = np.sqrt(mean_squared_error)
    return root_mean_squared_error
Solution = root_mean_square_error(tabulated_declination, cooper_declination)
print("Root Mean Square Error:", Solution)
```

Cooper's expression Percentage Error

```
# Percentage error
def calculate_percentage_error(tabulated_declination, cooper_declination):
    # Calculate percentage error for each pair of values
    errors = [(abs(tabulated_declination[i] - cooper_declination[i]) / tabulated_declination[i]) * 100
                for i in range(len(cooper_declination))]
    # Calculate the average percentage error
    average_error = sum(errors) / len(errors)
    return average_error
percentage_error = calculate_percentage_error(tabulated_declination, cooper_declination)
print(f"Average Percentage Error: {percentage_error:.2f}%")
```

Comparison of the four graphs

```
plt.figure()
plt.plot(days_in_year,tabulated_declination, "b", label="Igbal Tabulated")
plt.plot(days_in_year,spencer_declination,"r--",label="Spencer")
plt.plot(days_in_year,perrin_de_brichambaut, "k--",label =  "Perrin_de_Brichambauti")
plt.plot(days_in_year,cooper_declination, "y",label="Cooper")
plt.xlabel("Month")
plt.ylabel("Declination angle(${^\circ}$)")
plt.legend()
plt.grid()
plt.xticks(np.linspace(0,365,13)[:-1],('Jan','Feb','Mar','Apr','May','June','Jul','Aug','Sep','Oct',
'Nov','Dec') )
plt.savefig("tabulated_and_all_expressions.jpg")
plt.show()
```

Appendix 5: Computer Codes and Tables for Daylength

Calculating the Daylength

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from numpy import pi
from IPython.display import display
def daylength(latitude):
    # Create an array of each day of the year
    day_number = np.arange(1, 366)
    # Create an array of days from to 31 for use in tables
    days = np.arange(1, 32)
    # Calculate the day angle for each day of the year
    day_angle = (2 * np.pi * (day_number - 1) / 365)
    #We now use the spencer's expression to calculate the declination angle
    declination = ((0.006918-0.399912*np.cos(day_angle) +0.070257 * np.sin(day_angle)
     -0.006758*np.cos(2*day_angle) + 0.000907 * np.sin(2*day_angle) - 0.002697 *
      np.cos(3*day_angle) + 0.00148 * np.sin(3*day_angle)))
    # Calculate omega, the solar hour angle, using latitude and declination
    omega = -np.tan(declination) * np.tan(latitude * np.pi / 180)
    # We confine omega values within the acceptable range (-1.0 to 1.0)
    for i in range(len(omega)):
        if omega[i] >= 1.0:
            omega[i] = 1.0
```

```
    elif omega[i] <= -1.0:
        omega[i] = -1.0
# Calculate day length in hours using the solar hour angle
day_length = (2/15) * np.arccos(omega) * 180 / np.pi
# Plot the day length over the year
plt.figure()
plt.plot(day_number, day_length, label="Day Length", c="k")
plt.xlabel("month")
plt.ylabel("day_length $\mathrm{(hrs)}$")
plt.legend()
plt.grid()
plt.minorticks_on()
plt.xticks(np.linspace(0, 365, 13)[:-1], ('Jan', 'Feb', 'Mar','Apr', 'May', 'June', 'Jul', 'Aug', 'Sep',
                                          'Oct', 'Nov', 'Dec'))
plt.show()
#Partitioning the number of days in a year into months
January = day_length[0:31]
February = day_length[31:59]
March = day_length[59:90]
April = day_length[90:120]
May = day_length[120:151]
June = day_length[151:181]
July = day_length[181:212]
August = day_length[212:243]
```

```
  September = day_length[243:273]
  October = day_length[273:304]
  November = day_length[304:334]
  December = day_length[334:365]
  # Assigning zeroes to 29, 30 and 31st February, 30th of April, June, September and November
 # to fill up the gaps
  February=np.append(February,[0.00,0.00,0.00])
  April = np.append(April,[0.00])
  June = np.append(June,[0.00])
  September = np.append(September,[0.00])
  November = np.append(November,[0.00])
  #We now create the table of declination angle as obtained from Iqbal and Muhammad
 daylength_data={'Jan' : list(January), 'Feb' : list(February), 'Mar' : list(March),' Apr' : list(April),
  'May' : list(May), 'Jun' : list(June), 'Jul' : list(July),' Aug': list(August), 'Sep' : list(September),
  'October' : list(October), 'Nove' : list(November), 'Dec' : list(December)}
     # Create the data frame using the dictionary
  df_day_length = pd.DataFrame(data=daylength_data, index=days)
  # Save the dataframe to an excel sheet
  df_day_length.to_excel("daylength.xlsx", index=True)
  # Show results
  display(df_day_length.round(2))
```

For Lusaka

```
Lusaka = daylength(latitude = -15)
```

Daylengths for Lusaka

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	12.87	12.64	12.28	11.85	11.46	11.17	11.12	11.33	11.69	12.10	12.52	12.82
2	12.87	12.63	12.27	11.83	11.45	11.17	11.12	11.34	11.70	12.12	12.53	12.82
3	12.87	12.62	12.26	11.82	11.43	11.16	11.13	11.35	11.72	12.13	12.54	12.83
4	12.86	12.61	12.24	11.81	11.42	11.16	11.13	11.36	11.73	12.14	12.55	12.83
5	12.86	12.59	12.23	11.79	11.41	11.15	11.13	11.37	11.74	12.16	12.57	12.84
6	12.85	12.58	12.21	11.78	11.40	11.15	11.14	11.38	11.76	12.17	12.58	12.85
7	12.85	12.57	12.20	11.77	11.39	11.14	11.14	11.39	11.77	12.19	12.59	12.85
8	12.84	12.56	12.19	11.75	11.38	11.14	11.15	11.40	11.78	12.20	12.60	12.86
9	12.84	12.55	12.17	11.74	11.37	11.13	11.15	11.41	11.80	12.21	12.61	12.86
10	12.83	12.53	12.16	11.72	11.36	11.13	11.16	11.42	11.81	12.23	12.62	12.86
11	12.83	12.52	12.14	11.71	11.35	11.13	11.16	11.43	11.83	12.24	12.64	12.87
12	12.82	12.51	12.13	11.70	11.34	11.12	11.17	11.44	11.84	12.25	12.65	12.87

13	12.81	12.50	12.12	11.68	11.33	11.12	11.17	11.45	11.85	12.27	12.66	12.87
14	12.81	12.48	12.10	11.67	11.32	11.12	11.18	11.47	11.87	12.28	12.67	12.88
15	12.80	12.47	12.09	11.66	11.31	11.12	11.19	11.48	11.88	12.30	12.68	12.88
16	12.79	12.46	12.07	11.64	11.30	11.12	11.19	11.49	11.89	12.31	12.69	12.88
17	12.78	12.45	12.06	11.63	11.29	11.11	11.20	11.50	11.91	12.32	12.70	12.88
18	12.78	12.43	12.04	11.62	11.28	11.11	11.21	11.51	11.92	12.34	12.71	12.89
19	12.77	12.42	12.03	11.61	11.27	11.11	11.21	11.53	11.94	12.35	12.72	12.89
20	12.76	12.41	12.02	11.59	11.26	11.11	11.22	11.54	11.95	12.36	12.73	12.89
21	12.75	12.39	12.00	11.58	11.25	11.11	11.23	11.55	11.96	12.38	12.74	12.89
22	12.74	12.38	11.99	11.57	11.24	11.11	11.24	11.56	11.98	12.39	12.75	12.89
23	12.73	12.37	11.97	11.55	11.24	11.11	11.24	11.58	11.99	12.40	12.75	12.89
24	12.72	12.35	11.96	11.54	11.23	11.11	11.25	11.59	12.01	12.42	12.76	12.89
25	12.71	12.34	11.95	11.53	11.22	11.11	11.26	11.60	12.02	12.43	12.77	12.89
26	12.70	12.32	11.93	11.52	11.21	11.11	11.27	11.61	12.03	12.44	12.78	12.89

27	12.69	12.31	11.92	11.51	11.21	11.11	11.28	11.63	12.05	12.45	12.79	12.89
28	12.68	12.30	11.90	11.49	11.20	11.12	11.29	11.64	12.06	12.47	12.79	12.88
29	12.67		11.89	11.48	11.19	11.12	11.30	11.65	12.07	12.48	12.80	12.88
30	12.66		11.88	11.47	11.19	11.12	11.31	11.67	12.09	12.49	12.81	12.88
31	12.65		11.86		11.18		11.32	11.68		12.51		12.88

```
Greenland = daylength(latitude = 71.7069)
```

Daylengths for Greenland

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	0.00	2.56	8.70	13.73	19.10	24.00	24.00	23.31	15.62	10.84	5.35	0.00
2	0.00	2.92	8.87	13.89	19.32	24.00	24.00	22.52	15.45	10.68	5.13	0.00
3	0.00	3.24	9.04	14.05	19.55	24.00	24.00	22.01	15.28	10.52	4.90	0.00
4	0.00	3.54	9.21	14.21	19.79	24.00	24.00	21.60	15.12	10.36	4.67	0.00
5	0.00	3.82	9.38	14.38	20.04	24.00	24.00	21.25	14.96	10.20	4.43	0.00
6	0.00	4.09	9.54	14.54	20.30	24.00	24.00	20.93	14.79	10.04	4.18	0.00
7	0.00	4.35	9.71	14.70	20.57	24.00	24.00	20.63	14.63	9.88	3.92	0.00
8	0.00	4.59	9.88	14.87	20.87	24.00	24.00	20.36	14.47	9.71	3.65	0.00
9	0.00	4.83	10.04	15.03	21.18	24.00	24.00	20.10	14.31	9.55	3.36	0.00
10	0.00	5.06	10.20	15.20	21.53	24.00	24.00	19.85	14.15	9.38	3.05	0.00
11	0.00	5.28	10.37	15.37	21.93	24.00	24.00	19.61	13.99	9.22	2.71	0.00

12	0.00	5.50	10.53	15.53	22.42	24.00	24.00	19.38	13.83	9.05	2.33	0.00
13	0.00	5.71	10.69	15.70	23.13	24.00	24.00	19.15	13.67	8.88	1.89	0.00
14	0.00	5.92	10.85	15.87	24.00	24.00	24.00	18.94	13.51	8.72	1.31	0.00
15	0.00	6.12	11.01	16.05	24.00	24.00	24.00	18.73	13.36	8.55	0.00	0.00
16	0.00	6.32	11.17	16.22	24.00	24.00	24.00	18.52	13.20	8.37	0.00	0.00
17	0.00	6.52	11.33	16.39	24.00	24.00	24.00	18.32	13.04	8.20	0.00	0.00
18	0.00	6.71	11.49	16.57	24.00	24.00	24.00	18.12	12.88	8.03	0.00	0.00
19	0.00	6.90	11.65	16.75	24.00	24.00	24.00	17.93	12.73	7.85	0.00	0.00
20	0.00	7.09	11.81	16.93	24.00	24.00	24.00	17.74	12.57	7.68	0.00	0.00
21	0.00	7.28	11.97	17.11	24.00	24.00	24.00	17.55	12.41	7.50	0.00	0.00
22	0.00	7.46	12.13	17.29	24.00	24.00	24.00	17.36	12.26	7.31	0.00	0.00
23	0.00	7.64	12.29	17.48	24.00	24.00	24.00	17.18	12.10	7.13	0.00	0.00
24	0.00	7.82	12.45	17.67	24.00	24.00	24.00	17.00	11.94	6.95	0.00	0.00
25	0.00	8.00	12.61	17.86	24.00	24.00	24.00	16.82	11.79	6.76	0.00	0.00

26	0.00	8.18	12.77	18.06	24.00	24.00	24.00	16.64	11.63	6.57	0.00	0.00
27	0.00	8.35	12.93	18.26	24.00	24.00	24.00	16.47	11.47	6.37	0.00	0.00
28	0.00	8.53	13.09	18.46	24.00	24.00	24.00	16.29	11.31	6.18	0.00	0.00
29	0.98		13.25	18.67	24.00	24.00	24.00	16.12	11.16	5.97	0.00	0.00
30	1.67		13.41	18.88	24.00	24.00	24.00	15.95	11.00	5.77	0.00	0.00
31	2.16		13.57		24.00		24.00	15.78		5.56		0.00

```
DomeCharlie = daylength(latitude = -75.1000)
```

Daylengths for Dome Charlie

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	24.00	24.00	16.18	9.84	0.76	0.00	0.00	0.00	7.40	13.45	21.58	24.00
2	24.00	24.00	15.95	9.64	0.00	0.00	0.00	0.00	7.62	13.65	22.21	24.00
3	24.00	24.00	15.73	9.43	0.00	0.00	0.00	0.00	7.84	13.85	23.24	24.00
4	24.00	24.00	15.51	9.23	0.00	0.00	0.00	0.00	8.06	14.05	24.00	24.00
5	24.00	24.00	15.30	9.02	0.00	0.00	0.00	0.00	8.27	14.25	24.00	24.00
6	24.00	24.00	15.08	8.81	0.00	0.00	0.00	0.00	8.48	14.45	24.00	24.00
7	24.00	24.00	14.87	8.60	0.00	0.00	0.00	0.00	8.69	14.66	24.00	24.00
8	24.00	24.00	14.66	8.39	0.00	0.00	0.00	0.00	8.90	14.87	24.00	24.00
9	24.00	24.00	14.45	8.17	0.00	0.00	0.00	0.00	9.10	15.08	24.00	24.00
10	24.00	22.47	14.24	7.95	0.00	0.00	0.00	0.00	9.31	15.29	24.00	24.00
11	24.00	21.76	14.04	7.73	0.00	0.00	0.00	0.00	9.51	15.50	24.00	24.00

12	24.00	21.22	13.83	7.51	0.00	0.00	0.00	0.00	9.71	15.72	24.00	24.00
13	24.00	20.76	13.63	7.28	0.00	0.00	0.00	0.00	9.91	15.94	24.00	24.00
14	24.00	20.35	13.43	7.05	0.00	0.00	0.00	1.55	10.11	16.16	24.00	24.00
15	24.00	19.97	13.23	6.82	0.00	0.00	0.00	2.22	10.31	16.38	24.00	24.00
16	24.00	19.63	13.03	6.58	0.00	0.00	0.00	2.74	10.51	16.61	24.00	24.00
17	24.00	19.30	12.83	6.33	0.00	0.00	0.00	3.18	10.70	16.84	24.00	24.00
18	24.00	18.99	12.63	6.08	0.00	0.00	0.00	3.58	10.90	17.08	24.00	24.00
19	24.00	18.70	12.43	5.82	0.00	0.00	0.00	3.94	11.09	17.32	24.00	24.00
20	24.00	18.41	12.23	5.56	0.00	0.00	0.00	4.27	11.29	17.57	24.00	24.00
21	24.00	18.14	12.03	5.28	0.00	0.00	0.00	4.59	11.49	17.83	24.00	24.00
22	24.00	17.87	11.84	5.00	0.00	0.00	0.00	4.89	11.68	18.09	24.00	24.00
23	24.00	17.62	11.64	4.70	0.00	0.00	0.00	5.17	11.88	18.36	24.00	24.00
24	24.00	17.36	11.44	4.39	0.00	0.00	0.00	5.45	12.07	18.64	24.00	24.00
25	24.00	17.12	11.24	4.05	0.00	0.00	0.00	5.72	12.27	18.92	24.00	24.00

26	24.00	16.88	11.04	3.70	0.00	0.00	0.00	5.97	12.46	19.23	24.00	24.00
27	24.00	16.64	10.84	3.31	0.00	0.00	0.00	6.23	12.66	19.54	24.00	24.00
28	24.00	16.41	10.65	2.88	0.00	0.00	0.00	6.47	12.85	19.88	24.00	24.00
29	24.00		10.45	2.38	0.00	0.00	0.00	6.71	13.05	20.24	24.00	24.00
30	24.00		10.25	1.76	0.00	0.00	0.00	6.94	13.25	20.64	24.00	24.00
31	24.00		10.04		0.00		0.00	7.17		21.07		24

```
Nanyuki = daylength(latitude = 0.0074)
```

Daylengths for Nanyuki

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
2	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
3	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
4	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
5	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
6	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
7	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
8	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
9	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
10	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
11	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00

12	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
13	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
14	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
15	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
16	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
17	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
18	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
19	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
20	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
21	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
22	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
23	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
24	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
25	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00

26	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
27	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
28	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
29	12.00		12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
30	12.00		12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
31	12.00		12.00		12.00		12.00	12.00		12.00		12.00

Appendix 6: Computer Codes and Tables for Extraterrestrial Radiation

Calculating the Daily Extraterrestrial Radiation for Lusaka

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from IPython.display import display
#Create the function for calculating daily extraterrestrial radiation
def daily_extraterrestrial(latitude):
    day = np.arange(1,32)
    day_number = np.arange(1,366)
    day_angle = 2*np.pi*(day_number - 1)/365
    declination = ((0.006918-0.399912*np.cos(day_angle) +0.070257 * np.sin(day_angle)
     -0.006758*np.cos(2*day_angle) + 0.000907 * np.sin(2*day_angle) - 0.002697 *
      np.cos(3*day_angle) + 0.00148 * np.sin(3*day_angle)))
    eccentricity_factor = (1.000110 + 0.034221*np.cos(day_angle) + 0.001280*np.sin(day_angle)
                           + 0.000719*np.cos(2*day_angle) + 0.000077*np.sin(2*day_angle))
    omega=-np.tan(declination)*np.tan(latitude*np.pi/180)
    # Clip the values of w to be within the valid range [-1, 1]
    omega = np.clip(omega, -1.0, 1.0)
    sunrise_hour_angle = np.arccos(omega)
    daily_radiation =( (24*3600*1367/np.pi)*(eccentricity_factor) *
     ((np.cos(latitude*np.pi/180)*np.cos(declination)*np.sin(sunrise_hour_angle)
      +((sunrise_hour_angle)  *np.sin(latitude*np.pi/180)*np.sin(declination))))/1000000)
    maximum_value = np.max(daily_radiation)
```

```
print(f"Maximum Solar Radiation: {maximum_value:.2f} MJ/m^2")
minimum_value = np.min(daily_radiation)
print(f"Minimum Solar Radiation: {minimum_value:.2f} MJ/m^2")
#return daily_radiation
plt.figure()
plt.plot(day_number,daily_radiation,label="Extrterrestrial\
Radiation",c="k")
plt.xlabel("Month")
plt.ylabel("Daily_Radiation $\mathrm{MJ/m^2}$")
plt.legend()
plt.grid()
plt.minorticks_on()
plt.xticks(np.linspace(0,365,13)[:-1],('Jan','Feb','Mar','Apr','May','June','Jul','Aug','Sep','Oct',
                                       'Nov','Dec') )
plt.savefig("Lusaka_Extraterrestrial.jpg")
plt.show()
January = daily_radiation[0:31]
February = daily_radiation[31:59]
March = daily_radiation[59:90]
April = daily_radiation[90:120]
May = daily_radiation[120:151]
June = daily_radiation[151:181]
July = daily_radiation[181:212]
August = daily_radiation[212:243]
```

```
    September = daily_radiation[243:273]
    October = daily_radiation[273:304]
    November = daily_radiation[304:334]
    December = daily_radiation[334:365]


    February=np.append(February,[0.00,0.00,0.00])
    April = np.append(April,[0.00])
    June = np.append(June,[0.00])
    September = np.append(September,[0.00])
    November = np.append(November,[0.00])
    data = {'Jan': January, 'Feb': February, 'Mar': March, 'Apr': April, 'May': May, 'Jun': June, 'Jul':
        July, 'Aug': August, 'Sept': September, 'Oct': October, 'Nov': November, 'Dec': December}
    df_daily_radiation = pd.DataFrame(data= data, index=day)
    df_daily_radiation_rounded = df_daily_radiation.round(2)
    df_daily_radiation_rounded.to_excel("daily_radiation_rounded.xlsx",  index = True)
    display(df_daily_radiation)
    return

Lusaka_Radiation = daily_extraterrestrial(latitude = -15)
```

Daily Extraterrestrial Solar Radiation over Lusaka

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sept	Oct	Nov	Dec
1	41.01	40.55	38.81	35.14	30.80	27.37	26.67	28.95	33.04	36.95	39.64	40.74
2	41.00	40.51	38.72	35.00	30.66	27.30	26.70	29.07	33.18	37.06	39.70	40.76
3	41.00	40.47	38.63	34.86	30.52	27.23	26.73	29.18	33.32	37.17	39.76	40.78
4	41.00	40.43	38.53	34.72	30.39	27.17	26.76	29.30	33.46	37.28	39.81	40.79
5	41.00	40.39	38.44	34.57	30.25	27.10	26.80	29.42	33.59	37.39	39.86	40.81
6	41.00	40.35	38.34	34.43	30.12	27.05	26.85	29.54	33.73	37.50	39.92	40.82
7	40.99	40.30	38.23	34.28	29.99	26.99	26.89	29.66	33.87	37.60	39.97	40.84
8	40.99	40.26	38.13	34.14	29.86	26.94	26.94	29.78	34.01	37.71	40.01	40.85
9	40.98	40.21	38.03	33.99	29.73	26.89	26.99	29.91	34.15	37.81	40.06	40.86
10	40.97	40.16	37.92	33.85	29.60	26.84	27.05	30.04	34.29	37.91	40.10	40.88
11	40.97	40.11	37.81	33.70	29.48	26.80	27.10	30.16	34.42	38.00	40.15	40.89
12	40.96	40.05	37.70	33.55	29.35	26.76	27.17	30.29	34.56	38.10	40.19	40.90

13	40.95	40.00	37.58	33.41	29.23	26.73	27.23	30.42	34.69	38.20	40.23	40.91
14	40.94	39.94	37.47	33.26	29.11	26.69	27.30	30.55	34.83	38.29	40.27	40.92
15	40.93	39.88	37.35	33.11	29.00	26.66	27.37	30.69	34.96	38.38	40.31	40.93
16	40.92	39.81	37.24	32.96	28.88	26.64	27.44	30.82	35.09	38.47	40.34	40.94
17	40.9	39.75	37.11	32.82	28.77	26.62	27.52	30.95	35.23	38.55	40.38	40.95
18	40.89	39.68	36.99	32.67	28.66	26.60	27.59	31.09	35.36	38.64	40.41	40.95
19	40.87	39.62	36.87	32.52	28.55	26.58	27.67	31.23	35.49	38.72	40.44	40.96
20	40.85	39.54	36.75	32.37	28.44	26.57	27.76	31.36	35.62	38.81	40.47	40.97
21	40.84	39.47	36.62	32.23	28.34	26.56	27.84	31.50	35.74	38.89	40.50	40.97
22	40.82	39.40	36.49	32.08	28.24	26.56	27.93	31.64	35.87	38.96	40.53	40.98
23	40.80	39.32	36.36	31.94	28.14	26.56	28.03	31.78	36.00	39.04	40.56	40.98
24	40.77	39.24	36.23	31.79	28.04	26.56	28.12	31.91	36.12	39.11	40.59	40.99
25	40.75	39.16	36.10	31.65	27.95	26.56	28.22	32.05	36.24	39.19	40.61	40.99
26	40.73	39.08	35.97	31.50	27.86	26.57	28.31	32.19	36.36	39.26	40.63	41.00

27	40.70	38.99	35.83	31.36	27.77	26.58	28.42	32.33	36.48	39.33	40.66	41.00
28	40.67	38.90	35.69	31.22	27.68	26.60	28.52	32.47	36.60	39.39	40.68	41.00
29	40.64		35.56	31.08	27.60	26.62	28.62	32.61	36.72	39.46	40.70	41.00
30	40.61		35.42	30.94	27.52	26.64	28.73	32.75	36.84	39.52	40.72	41.00
31	40.58		35.28		27.44		28.84	32.89		39.58		41.01

Monthly Average Daily Extraterrestrial Solar Radiation at different Latitudes

```
import numpy as np
latitude = np.linspace(90, -90, 37)
#The following days were picked to represent the average radiation for each month
day_number = np.array([17, 45, 74, 105, 135, 161, 199, 230, 261, 292, 322, 347])
daily_radiation = []
for lat in latitude:
    day_angle = 2 * np.pi * (day_number - 1) / 365
    declination = (
        0.006918
        - 0.399912 * np.cos(day_angle)
        + 0.070257 * np.sin(day_angle)
        - 0.006758 * np.cos(2 * day_angle)
        + 0.000907 * np.sin(2 * day_angle)
        - 0.002697 * np.cos(3 * day_angle)
        + 0.00148 * np.sin(3 * day_angle)
    )
    eccentricity_factor = (1.000110 + 0.034221 * np.cos(day_angle) + 0.001280 *
        np.sin(day_angle) + 0.000719 * np.cos(2 * day_angle) + 0.000077 * np.sin(2 * day_angle))
    omega = -np.tan(declination) * np.tan(lat * np.pi / 180)
    omega = np.clip(omega, -1.0, 1.0)
    sunrise_hour_angle = np.arccos(omega)
    radiation = ((24 * 3600 * 1367 / np.pi) * eccentricity_factor * (np.cos(lat * np.pi / 180) *
        np.cos(declination) * np.sin(sunrise_hour_angle)+ sunrise_hour_angle *
```

```
        np.sin(lat * np.pi / 180) * np.sin(declination)) / 1000000)
    daily_radiation.append(radiation)
row_labels = latitude
df_average_daily_radiation = pd.DataFrame(data = daily_radiation, index = row_labels, columns=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
df_average_daily_radiation.index.name = 'Latitude'
df_average_daily_radiation_rounded = df_average_daily_radiation.round(2)
df_average_daily_radiation_rounded.to_excel('Monthly_Average.xlsx', index = True)
df_average_daily_radiation_rounded
```

Monthly Average Daily Extraterrestrial Solar Radiation at different latitudes

Latitude	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
90	0.00	0.00	0.00	19.32	36.98	44.66	41.28	26.61	4.47	0.00	0.00	0.00
85	0.00	0.00	1.18	19.25	36.84	44.49	41.12	26.51	5.78	0.00	0.00	0.00
80	0.00	0.00	4.29	19.10	36.42	43.98	40.65	26.21	8.81	0.03	0.00	0.00
75	0.00	0.38	7.50	20.78	35.72	43.14	39.87	26.04	11.88	2.03	0.00	0.00
70	0.00	2.56	10.70	23.03	34.97	41.97	38.79	27.41	14.89	4.79	0.14	0.00
65	1.12	5.32	13.82	25.33	35.55	40.85	38.36	29.14	17.80	7.78	1.94	0.36
60	3.43	8.30	16.85	27.56	36.51	40.88	38.82	30.89	20.57	10.82	4.46	2.28
55	6.17	11.36	19.75	29.64	37.50	41.18	39.42	32.55	23.20	13.86	7.29	4.82
50	9.12	14.42	22.50	31.55	38.40	41.49	39.98	34.05	25.65	16.84	10.28	7.67
45	12.17	17.44	25.09	33.24	39.12	41.68	40.40	35.35	27.90	19.72	13.32	10.67
40	15.26	20.37	27.49	34.70	39.65	41.70	40.63	36.43	29.95	22.47	16.36	13.75
35	18.32	23.18	29.68	35.92	39.94	41.51	40.65	37.26	31.76	25.08	19.36	16.85

30	21.32	25.85	31.64	36.88	39.99	41.10	40.43	37.84	33.34	27.50	22.27	19.91
25	24.23	28.34	33.36	37.56	39.77	40.44	39.96	38.16	34.66	29.74	25.06	22.90
20	27.00	30.64	34.83	37.98	39.30	39.55	39.25	38.2	35.72	31.75	27.71	25.78
15	29.62	32.73	36.04	38.11	38.57	38.40	38.28	37.98	36.51	33.53	30.19	28.53
10	32.06	34.58	36.97	37.96	37.57	37.02	37.06	37.48	37.02	35.07	32.47	31.12
5	34.30	36.19	37.62	37.54	36.32	35.40	35.60	36.71	37.25	36.35	34.55	33.52
0	36.32	37.54	37.99	36.83	34.83	33.56	33.91	35.68	37.20	37.36	36.39	35.71
-5	38.10	38.62	38.07	35.85	33.10	31.51	32.01	34.39	36.87	38.10	37.99	37.68
-10	39.63	39.42	37.85	34.61	31.15	29.26	29.89	32.86	36.25	38.55	39.33	39.42
-15	40.90	39.94	37.35	33.11	29.00	26.84	27.59	31.09	35.36	38.72	40.41	40.91
-20	41.91	40.17	36.57	31.37	26.65	24.27	25.13	29.10	34.20	38.61	41.22	42.14
-25	42.65	40.12	35.51	29.40	24.15	21.57	22.52	26.91	32.78	38.21	41.75	43.11
-30	43.11	39.78	34.18	27.21	21.49	18.77	19.79	24.54	31.11	37.53	42.02	43.83
-35	43.32	39.16	32.59	24.83	18.73	15.90	16.97	22.00	29.20	36.58	42.01	44.28

-40	43.27	38.28	30.75	22.28	15.88	12.99	14.10	19.32	27.08	35.36	41.75	44.50
-45	42.98	37.14	28.68	19.58	12.97	10.10	11.21	16.53	24.75	33.90	41.24	44.50
-50	42.50	35.76	26.40	16.74	10.07	7.28	8.36	13.67	22.23	32.20	40.53	44.31
-55	41.86	34.18	23.91	13.81	7.21	4.60	5.61	10.75	19.54	30.28	39.64	44.00
-60	41.17	32.43	21.24	10.82	4.48	2.20	3.07	7.85	16.71	28.19	38.66	43.71
-65	40.62	30.57	18.42	7.82	2.03	0.37	0.95	5.02	13.75	25.95	37.73	43.71
-70	40.95	28.74	15.47	4.87	0.22	0.00	0.00	2.40	10.70	23.63	37.24	44.95
-75	42.09	27.29	12.41	2.12	0.00	0.00	0.00	0.34	7.57	21.39	38.15	46.20
-80	42.91	27.44	9.29	0.07	0.00	0.00	0.00	0.00	4.41	19.78	38.89	47.11
-85	43.41	27.75	6.24	0.00	0.00	0.00	0.00	0.00	1.33	19.97	39.34	47.65
-90	43.58	27.86	5.08	0.00	0.00	0.00	0.00	0.00	0.00	20.05	39.49	47.83

Graphs for Monthly Average Daily Extraterrestrial Solar Radiation at different latitudes

```
import numpy as np
import matplotlib.pyplot as plt
latitude = np.linspace(-90,90)
def extraterrestrial(day_number):
    day_angle = 2*np.pi*(day_number - 1)/365
    #latitude = np.linspace(0,90)
        declination = (
        0.006918
        - 0.399912 * np.cos(day_angle)
        + 0.070257 * np.sin(day_angle)
        - 0.006758 * np.cos(2 * day_angle)
        + 0.000907 * np.sin(2 * day_angle)
        - 0.002697 * np.cos(3 * day_angle)
        + 0.00148 * np.sin(3 * day_angle)
    )
    eccentricity_factor = (1.000110 + 0.034221 * np.cos(day_angle) + 0.001280 *
          np.sin(day_angle) + 0.000719 * np.cos(2 * day_angle) + 0.000077 * np.sin(2 * day_angle))
    omega=-np.tan(declination)*np.tan(latitude*np.pi/180)

    # Clip the values of omega to be within the valid range [-1, 1]
    omega = np.clip(omega, -1.0, 1.0)
    sunrise_hour_angle = np.arccos(omega)
   Daily_Radiation = (24*3600*1367/np.pi)*(eccentricity_factor)*  ((np.cos(latitude*np.pi/180) *
```

```
                    np.cos(declination)*np.sin(sunrise_hour_angle)+((sunrise_hour_angle)*
                    np.sin(latitude*np.pi/180) * np.sin(declination))))/1000000
    return Daily_Radiation
```

Calling on the function to plot average monthly extraterrestrial radiation at different latitudes for the month of January.

```
Daily_Radiation_January= extraterrestrial(day_number=17)
plt.figure()
plt.plot(latitude, Daily_Radiation_January, c ="k")
plt.xlabel("Latitude ($^{o}$)")
plt.ylabel("Daily Radiation (MJ/$m^{2}$)")
plt.grid()
plt.show()
```

Monthly Average Daily Extraterrestrial Solar Radiation at different latitudes for all the months

```
plt.figure()
# List of month names
months = ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"]
# Iterate over each month and plot the curve
for month in months:
    # Replace "Daily_Radiation_{month}_south" with your actual variable names
    daily_radiation = globals()[f"Daily_Radiation_{month}"]
    plt.plot(latitude, daily_radiation)
    # Find the index corresponding to the latitude value closest to 45 degrees
    index_60_degrees = np.abs(latitude - 60).argmin()
```

```
    # Add the label at latitude 45 for each curve
    plt.text(latitude[index_60_degrees], daily_radiation[index_60_degrees], month, ha='center', va='bottom')
plt.xlabel("Latitude ($^{o}$)")
plt.ylabel("Daily Radiation (MJ/$m^{2}$)")
plt.grid()
plt.show()
```

Appendix 7: Computer Codes for Reflection and Refraction, and Reflectance and Transmittance

Code for Interactive Plots of Reflection and Refraction at Dielectric Interface

```
import numpy as np
import matplotlib.pyplot as plt
import Ipywidgets as widgets
from numpy import pi
# Create input widgets for the refractive indices
n_i = widgets.FloatText(value=0, description='Index 1:')
n_r = widgets.FloatText(value=0, description='Index 2:')
#n_i is the refractive index of the first medium, n_r is the refractive index of the transmitted medium
# Create output widget for displaying the plot and zero reflectance angle
result_output = widgets.Output()
# Generate an array of angles
thetai = np.linspace(0, 90, 500)
theta = thetai * pi / 180
# Function to calculate reflectance values for both inputs
def calculate_results():
    result_i = np.zeros_like(thetai)
    result_r = np.zeros_like(thetai)
    r_average = np.zeros_like(thetai)
    # Checking for the valid values of refractive indices
    valid_values_i = n_r.value**2 - n_i.value**2 * np.sin(theta)**2 >= 0
    valid_values_r = n_r.value**2 - n_i.value**2 * np.sin(theta)**2 >= 0
```

```
    # Update only valid values
    result_i[valid_values_i] = ((pow(n_r.value, 2) *
    np.cos(theta[valid_values_i])  - n_i.value *
    np.sqrt(n_r.value ** 2 - n_i.value ** 2 *
    np.sin(theta[valid_values_i]) ** 2))/ (n_r.value ** 2 *
    np.cos(theta[valid_values_i]) + n_i.value * np.sqrt(n_r.value ** 2 -
    n_i.value ** 2 * np.sin(theta[valid_values_i]) ** 2))
    ) ** 2
    result_r[valid_values_r] = ((n_i.value * np.cos(theta[valid_values_r]) –
        np.sqrt(n_r.value**2-n_i.value**2*np.sin(theta[valid_values_r])**2)) /
        (n_i.value * np.cos(theta[valid_values_r])  +  np.sqrt(n_r.value**2 - n_i.value**2 *
        np.sin(theta[valid_values_r])**2)) ) ** 2
    # Update r_average only where both result_i and result_r are valid
    r_average[valid_values_i & valid_values_r] = 0.5 *  (result_i[valid_values_i & valid_values_r]
                                        + result_r[valid_values_i & valid_values_r])
    # Find the index where reflectance is closest to zero for input 1
    zero_reflectance_index = np.argmin(np.abs(result_i))
    return result_i, result_r, r_average, zero_reflectance_index
# Function to update the plot and display zero reflectance angle
def update_plot(b):
    with result_output:
        result_output.clear_output(wait=True)
        result_i, result_r, r_average, zero_reflectance_index =  calculate_results()
        # Plot the results for both inputs on the same graph
```

```
        plt.figure()
        plt.plot(thetai, result_i, label="Parallel")
        plt.plot(thetai, result_r, label="perpendicular")
        plt.plot(thetai, r_average, label="r_average")
        plt.xlabel("$\Theta_i$")
        plt.ylabel("Reflectance")
        plt.legend()
        plt.grid()
        plt.savefig("Reflectance_Comparison.jpg")
        plt.show()
        # Display the angle where reflectance is closest to zero for input 1
        zero_reflectance_angle = thetai[zero_reflectance_index]
       print(f"The angle where reflectance is closest to zero for  parallel is: {zero_reflectance_angle}
                    degrees")
# Create buttons for plotting and resetting
plot_button = widgets.Button(description="Plot")
reset_button = widgets.Button(description="Reset")
# Define button click functions
plot_button.on_click(update_plot)
reset_button.on_click(lambda b: result_output.clear_output())
# Display input widgets, buttons, and output widget
display(widgets.HBox([n_i, n_r, plot_button, reset_button]), result_output)
```

Code for Reflectance and Transmittance

```
import numpy as np
import matplotlib.pyplot as plt
import Ipywidgets as widgets
from numpy import pi
alpha = 1
# Create input widgets
n_i = widgets.FloatText(value=0, description='Index 1:')
n_r = widgets.FloatText(value=0, description='Index 2:')
# Create output widget
result_output = widgets.Output()
thetai = np.linspace(0, 90, 500)
theta = thetai * pi / 180
# Function to calculate results for both inputs
def calculate_results():
    result_i = (
        (pow(n_r.value, 2) * np.cos(theta) - n_i.value * np.sqrt(n_r.value
         ** 2 - n_i.value ** 2 * np.sin(theta) ** 2))/ (n_r.value ** 2 *
         np.cos(theta) + n_i.value * np.sqrt(n_r.value ** 2 - n_i.value
         ** 2 * np.sin(theta) ** 2))) ** 2
    result_r = (
        (n_i.value*np.cos(theta)-np.sqrt(n_r.value**2-n_i.value**2
       *np.sin(theta)**2))/(n_i.value*np.cos(theta)+np.sqrt(n_r.value**2-
        n_i.value**2*np.sin(theta)**2))) ** 2
```

```
    T_average = (alpha*0.5)*(((1-result_i)/(1+result_i))+((1-result_r) /(1+result_r)))
    R_average = 1-T_average
    return T_average, R_average
# Function to find the angles where reflectance is zero for input 1
def find_zero_reflectance_angle(result_array, thetai_array):
    zero_reflectance_angles = thetai_array[result_array == 0]
    return zero_reflectance_angles
# Function to update the plot when the "Plot" button is clicked
def update_plot(b):
    with result_output:
        result_output.clear_output(wait=True)
        T_average, R_average = calculate_results()
        # Plot the results for both inputs on the same graph
        plt.figure()
        plt.plot(thetai, T_average, label="T-average")
        plt.plot(thetai, R_average, label="R_average")
        plt.xlabel("$\Theta_i$")
        plt.ylabel("Reflectance")
        plt.legend()
        plt.grid()
        plt.show()
# Function to reset the input values and clear the output
def reset_values(b):
    n_i.value = 0
```

```
    n_r.value = 0
    result_output.clear_output()
# Create buttons for plotting and resetting
plot_button = widgets.Button(description="Plot")
reset_button = widgets.Button(description="Reset")
# Define button click functions
plot_button.on_click(update_plot)
reset_button.on_click(reset_values)
# Display widgets
display(widgets.HBox([n_i, n_r, plot_button, reset_button]), result_output)
```